Omega ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Omega

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

Just-in-time scheduling with two competing agents on unrelated parallel machines ☆

Yunqiang Yin [a,*], Shuenn-Ren Cheng [b], T.C.E. Cheng [c], Du-Juan Wang [d,*], Chin-Chia Wu [e]

[a] Faculty of Science, Kunming University of Science and Technology, Kunming 650093, China
[b] Graduate Institute of Business Administration, Cheng Shiu University, Kaohsiung County, Taiwan
[c] Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
[d] School of Management Science and Engineering, Dalian University of Technology, Dalian 116023, China
[e] Department of Statistics, Feng Chia University, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 1 August 2014
Accepted 3 September 2015

Keywords:
Scheduling
Two agents
Unrelated parallel machines
Just-in-time scheduling
FPTAS

ABSTRACT

This paper considers two-agent just-in-time scheduling where agents A and B have to share m unrelated parallel machines for processing their jobs. The objective of agent A is to maximize the weighted number of its just-in-time jobs that are completed exactly on their due dates, while the objective of agent B is either to maximize its maximum gain (income) from its just-in-time jobs or to maximize the weighted number of its just-in-time jobs. We provide a bicriterion analysis of the problem, which seek to find the Pareto-optimal solutions for each combination of the two agents' criteria. When the number of machines is part of the problem instance, both the addressed problems are *NP*-hard in the strong sense. When the number of machines is fixed, we show that the problem of maximizing agent A's weighted number of just-in-time jobs while maximizing agent B's maximum gain can be solved in polynomial time, whereas the problem of maximizing both agents' weighted numbers of just-in-time jobs is $\mathcal{NP}$-hard. For the latter problem, we also provide a pseudo-polynomial-time solution algorithm, establishing that it is $\mathcal{NP}$-hard in the ordinary sense, and show that it admits a fully polynomial-time approximation scheme (FPTAS) for finding an approximate Pareto solution.

1. Introduction

In classical just-in-time scheduling, the objective is to minimize the earliness–tardiness cost of completing jobs with respect to their due dates. Most papers in this area consider the objective of minimizing the sum of the earliness and tardiness penalties (see, e.g., [9,24,27,28]). In some situations, the earliness and tardiness penalties depend on whether the jobs are early or tardy, rather than how early or how late they are. Lann and Mosheiov [15] introduce a new objective of minimizing the weighted number of jobs that are early or tardy, which corresponds to maximizing the weighted number of jobs that are completed exactly on their due dates. We refer to any scheduling problem with the objective of maximizing the weighted number of just-in-time jobs as a just-in-time scheduling problem.

Lann and Mosheiov's study has recently received growing attention from the scheduling research community and just-in-time scheduling has been studied in various machine settings. For the single-machine case, Lann and Mosheiov [15] show that the just-in-time scheduling problem is solvable in $O(n^2)$ time, where n is the number of jobs. For the two-machine flowshop case, Choi and Yoon [8] prove that it is $\mathcal{NP}$-hard, but they leave an open question whether the problem is $\mathcal{NP}$-hard in the ordinary sense or in the strong sense. In addition, they show that the unweighted version of the problem can be solved in $O(n^4)$ time for the two parallel-machine case and is $\mathcal{NP}$-hard in the strong sense for the three parallel-machine case. Shabtay and Bensoussan [20] show that the open problem left in Choi and Yoon [8] is $\mathcal{NP}$-hard in the ordinary sense by developing a pseudo-polynomial-time algorithm and a fully polynomial-time approximation scheme (FPTAS) for the problem. Elalouf et al. [10] suggest another pseudo-polynomial-time algorithm for the same problem, which can be converted into a new FPTAS that reduces Shabtay and Bensoussan's complexity result. Shabtay [19] studies the just-in-time problem in the flowshop setting under four different scenarios. For each scenario, he either presents a polynomial-time algorithm or develops an efficient pseudo-polynomial-time algorithm. Shabtay et al. [21] address a two-machine flowshop scheduling problem where the job processing time is controllable by varying the allocation of a resource to the job operations. They adopt a

☆This manuscript was processed by Associate Editor Tkindt

* Corresponding authors.
E-mail addresses: yinyunqiang@126.com (Y. Yin), wangdujuan@dlut.edu.cn (D.-J. Wang).

http://dx.doi.org/10.1016/j.omega.2015.09.010
0305-0483/

bicriterion analysis of the problem in which the first objective is to maximize the weighted number of just-in-time jobs while the second objective is to minimize the total resource consumption cost. They develop a pseudo-polynomial-time algorithm for the problem and convert it into a two-dimensional FPTAS. In the parallel-machine setting, Carlisle and Lloyd [6] consider the unweighted version of the just-in-time scheduling problem on m identical parallel machines and show that the problem can be solved in $O(n \log n)$ time. Other solution algorithms for the same problem can be found in Čepek and Sung [7], Frank [11], Yannakakis and Gavril [26], and Hsiao et al. [13]. Arkin and Silverberg [2] develop an $O(n^2 \log n)$ time solution algorithm for the weighted case on m identical parallel machines by converting the problem into a minimum cost flow problem. Bouzina and Emmonss [5], and Carlisle and Lloyd [6] present more efficient minimum cost flow algorithms with an $O(mn \log n)$ running time for the same problem by modelling it on a network that has only $O(n)$ arcs. Kovalyov et al. [14] show that the just-in-time scheduling problem on unrelated parallel machines is equivalent to that of maximizing the weighted m legally colourable vertices in a given interval graph. Both Arkin and Silverberg [2], and Sung and Vlach [23] prove that the just-in-time scheduling problem on m unrelated parallel machines can be solved in $O(mn^{m+1})$ time, which is polynomial when m is fixed. However, when m is arbitrary, they show that the problem becomes $\mathcal{NP}$-hard in the strong sense. Leyvand et al. [16] consider just-in-time scheduling on a set of m machines with controllable processing times, where the objectives are to maximize the weighted number of just-in-time jobs and to minimize the total resource allocation cost. They consider four different models for treating the two criteria. For each model, they either provide a polynomial-time solution algorithm or develop a pseudo-polynomial-time solution algorithm and an FPTAS.

All the above papers focus on the traditional case of just-in-time scheduling with a single agent. In recent years researchers have increasingly considered scheduling with multiple competing agents, which was initially investigated by Agnetis et al. [1] and Baker and Smith [3]. In this case, multiple agents need to process their own sets of jobs, competing for the use of a common resource. Each agent wants to optimize a certain objective function, which depends on the completion times of its jobs only. Variants of the scheduling problem with multiple agents have found many applications in areas such as manufacturing, supply chain management, telecommunication services, project scheduling, etc. A recent survey of multi-agent scheduling research is given in Perez–Gonzalez and Framinan [17]. With a view to modelling a realistic production system, this paper combines the two sub-fields into a unified framework. Specifically, we focus on the innovative just-in-time scheduling model on unrelated parallel machines in the two-agent setting. The purpose of this paper is twofold. One is to investigate this unexplored scheduling model. Another is to ascertain the computational complexity status and provide solution procedures, if viable, for the problems under consideration.

The rest of the paper is organized as follows: In Section 2 we formulate the problem and present a common property of the optimal schedules for the two problems under consideration. In Section 3 we show that the Pareto-optimization problem with a fixed number of machines where agent A's objective is to maximize the weighted number of its just-in-time jobs while agent B's objective is to maximize its maximum gain can be solved in polynomial time. In Section 4 we show that the Pareto-optimization problem with a fixed number of machines where both agents' objectives are to maximize their weighted numbers of just-in-time jobs is $\mathcal{NP}$-hard in the ordinary sense by developing a pseudo-polynomial-time algorithm for the problem and we convert the algorithm into an FPTAS. In the last section we provide some concluding remarks and suggest topics for future research.

2. Problem formulation

We formally describe the problem under study as follows: There are two competing agents (called agent A and agent B, respectively) that have to schedule two families of independent and non-preemptive jobs on m unrelated parallel machines $M_1, M_2, \ldots, M_m$. Agent A has to perform the job set $J^A = \{J_1^A, J_2^A, \ldots, J_{n_A}^A\}$, while agent B has to perform the job set $J^B = \{J_1^B, J_2^B, \ldots, J_{n_B}^B\}$. We call the jobs of agents A and B the A-jobs and B-jobs, respectively. All the jobs are available for processing from time zero onwards. Let $X \in \{A, B\}$ and let $n = n_A + n_B$ denote the total number of jobs. Denote d_j^X as the due date of job J_j^X, w_j^X as the gain (income) from completing job J_j^X just-in-time (i.e., exactly at time d_j^X), and p_{ij}^X as the processing time of job J_j^X on machine M_i for $i = 1, \ldots, m$ and $j = 1, \ldots, n_X$. The jobs that are completed exactly on their due dates in some schedule are called just-in-time jobs. We assume, without loss of generality, that all the d_j^X, w_j^X, and p_{ij}^X values are positive integers, and let $W^A = \sum_{J_k^A \in J^A} w_k^A$ and $W^B = \sum_{J_k^B \in J^B} w_k^B$.

For any given solution, let E_i be the set of just-in-time jobs allocated to machine M_i for $i = 1, \ldots, m$, with $E = E_1 \cup E_2 \cup \cdots \cup E_m$, and let $T = (J^A \cup J^B) \backslash E$ be the set of the other jobs. A partition of set $J^A \cup J^B$ into two disjointed subsets E and T is considered to be a feasible partition (or a feasible schedule) if it is possible to schedule the jobs belonging to set E on the m unrelated parallel machines such that they are all completed just in time. Following Lann and Mosheiov [15], in a feasible schedule, it is assumed that the jobs in T need not be executed, which means that they are rejected. We also denote by E^A and E^B the sets of just-in-time A-jobs and B-jobs, respectively.

Each agent wants to optimize a certain objective function depending on the completion times of its jobs only. Specifically, agent A wants to maximize $Q^A(S) = \sum_{J_k^A \in E^A} w_k^A$ (the weighted number of just-in-time A-jobs, i.e., the total gain from completing the jobs in set E^A just-in -time), while agent B wants to maximize $Q^B(S) = \max_{J_k^B \in E^B} w_k^B$ (the maximum gain from completing the jobs in set E^B just-in-time) or to maximize $Q^B(S) = \sum_{J_k^B \in E^B} w_k^B$ (the weighted number of just-in-time B-jobs, i.e., the total gain from completing the jobs in set E^B just-in-time). Since increasing the objective value of agent A will decrease the objective value of agent B, and vice versa, we need to consider the trade-off between the two objective functions carefully to achieve the best scheduling outcome. For such kind of bicriterion problem, we focus on finding the set of all the Pareto-optimal schedules (points) (Q^A, Q^B), where a schedule S with $Q^A = Q^A(S)$ and $Q^B = Q^B(S)$ is called Pareto-optimal (or efficient) if there does not exist another schedule S' such that $Q^A(S') \geq Q^A(S)$ and $Q^B(S') \geq Q^B(S)$ with at least one of these inequalities being strict. Using the three-field notation proposed by Graham et al. [12] and extended to multicriteria scheduling problems by T'kindt and Billaut [25], we denote the problems under consideration by $Rm||(\sum_{J_k^A \in E^A} w_k^A, \max_{J_k^B \in E^B} w_k^B)$ and $Rm||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$, respectively, when the number of machines m is fixed. Note that the criterion in the classical three-field notation is to be minimized, but in this paper the criterion is to be maximized.

Sung and Vlach [23] have shown that the just-in-time scheduling problem on unrelated parallel machine is NP-hard in the strong sense if the number of machines is part of the problem instance. In fact, if the due dates of the B-jobs are made very large, our problems reduce to their problem, so our problems are NP-hard in the strong sense when the number of machines is part of the problem instance, too. Hence, in what follows, we focus only on the case where the number of the machines is fixed.

The following lemma provides an easy-to-prove property for the problems under consideration.

Please cite this article as: Yin Y, et al. Just-in-time scheduling with two competing agents on unrelated parallel machines. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.09.010

Lemma 2.1. *There exists an optimal solution in which the jobs in the just-in-time set E are sequenced in the earliest due-date (EDD) order.*

Let $J^X_{i[j]}$, $i=1,\ldots,m, j=1,\ldots,|E_i|$, be the jth just-in-time job on machine M_i, where $X=A$ means that the jth job belongs to agent A while $X=B$ means that the jth job belongs to agent B. The following inequality provides a necessary condition for including job $J^X_{i[j]}$ in E_i right after job $J^Y_{i[j-1]}$:

$$p^X_{i[j]} \le d^X_{[j]} - d^Y_{[j-1]}, \tag{1}$$

where $d^X_{i[0]}=0$ and $X,Y\in\{A,B\}$, because otherwise job $J^X_{i[j]}$ cannot be completed just-in-time on machine M_i, given that job $J^Y_{i[j-1]}$ is completed just-in-time on the same machine. Without loss of generality, we may assume that $\min_{i=1,\ldots,m} p^Z_{ij} \le d^Z_j$ for $j=1,\ldots,n_Z$, where $Z\in\{A,B\}$. Otherwise, job J^Z_j cannot be completed just-in-time in any schedule, so it can be eliminated from the problem.

3. The $Rm||(\sum_{J^A_k\in E^A} w^A_k, \max_{J^B_k\in E^B} w^B_k)$ problem

In this section we consider the $Rm||(\sum_{J^A_k\in E^A} w^A_k, \max_{J^B_k\in E^B} w^B_k)$ problem, in which agent A wants to maximize the weighted number of its just-in-time jobs while agent B wants to maximize its maximum gain.

It is evident that there exists an optimal schedule for the $Rm||(\sum_{J^A_k\in E^A} w^A_k, \max_{J^B_k\in E^B} w^B_k)$ problem in which there is exactly one just-in-time B-job. However, we do not know which B-job is completed just-in-time and on which machine it is processed in the optimal schedule. So we need to enumerate all the possible cases. For each assigned just-in-time B-job $J_h{}^B$, $h=1,\ldots,n_B$, on some machine M_t, $t=1,\ldots,m$, there is an unavailability interval $[d^B_h-p^B_{th}, d^B_h)$ on machine M_t, so the only remaining question is how to schedule the A-jobs on the m unrelated parallel machines in the EDD order. However, both the algorithms developed in Arkin and Silverberg [2], and Sung and Vlach [23] cannot be applied for this case. In what follows, we design a dynamic programming (DP) algorithm to solve it. Temporarily assume, in this section, that the jobs in J^A are re-indexed in the EDD order.

Let $(\mathfrak{F}^{(j)},\mathcal{V}^A)_{(t,h)}$ be a state corresponding to a feasible partial schedule for the first j jobs $\{J^A_1,\ldots,J^A_j\}$, given that the sole just-in-time B-job $J_h{}^B$ is assigned to machine M_t in the final optimal schedule, in which the following information is recorded:

- $\mathfrak{F}^{(j)}=(k_1,\ldots,k_m)$: k_i $(0\le k_i\le j)$ denotes that the last just-in-time A-job on machine M_i is $J^A_{k_i}$ with $k_p\ne k_q$ if $k_p\ne 0$ and $p\ne q$ (i.e., no job can be processed on two different machines), and $k_i=0$ meaning that no job is scheduled on machine M_i in the partial schedule;
- $\mathcal{V}^A$: the maximum weighted number of just-in-time A-jobs in the partial schedule.

Before describing the DP algorithm in detail, let us first develop an elimination property.

Lemma 3.1. *For any two states $(\mathfrak{F}^{(j)},\mathcal{V}^A_1)_{(t,h)}$ and $(\mathfrak{F}^{(j)},\mathcal{V}^A_2)_{(t,h)}$ with $\mathcal{V}^A_1\ge\mathcal{V}^A_2$, the second state can be eliminated.*

Proof. Let S_1 and S_2 be two partial schedules corresponding to the states $(\mathfrak{F}^{(j)},\mathcal{V}^A_1)_{(t,h)}$ and $(\mathfrak{F}^{(j)},\mathcal{V}^A_2)_{(t,h)}$, respectively. And let $N=\{J^A_{k_1},J^A_{k_2},\ldots,J^A_{k_l}\}$ be a set of just-in-time A-jobs that are added to the partial schedule S_2 with $k_l>\cdots>k_2>k_1>j$ so as to create a feasible schedule $\widehat{S}_2$. In the resulting feasible schedule $\widehat{S}_2$, the objective value of agent A is given as follows:

$$Q^A(\widehat{S}_2)=\mathcal{V}^A_2+\sum_{J^A_{k_r}\in N} w_{k_r}.$$

Since the first coordinates in the two states remain the same, set N can also be added to the partial schedule S_1 to form a feasible schedule $\widehat{S}_1$. In the resulting feasible schedule $\widehat{S}_1$, the objective value of agent A is given as follows:

$$Q^A(\widehat{S}_1)=\mathcal{V}^A_1+\sum_{J^A_{k_r}\in N} w_{k_r}.$$

It follows from $\mathcal{V}^A_1\ge\mathcal{V}^A_2$ that $Q^A(\widehat{S}_1)\ge Q^A(\widehat{S}_2)$. Therefore, the partial schedule S_1 dominates S_2, and the result follows. □

Our algorithm uses Lemmas 2.1 and 3.1 to solve the problem in pseudo-polynomial time by finding the entire set of non-eliminated (partial) solutions. This is achieved by dynamically updating a set $\mathcal{L}^{(t,h,j)}$ of solution states $(\mathfrak{F}^{(j)},\mathcal{V}^A)_{(t,h)}$ from $\mathcal{L}^{(t,h,j-1)}$ for $t=1,\ldots,m$, $h=1,\ldots,n_B$, and $j=1,\ldots,n_A$. We initialize the algorithm by setting $(\mathfrak{F}^{(0)}=(0,\ldots,0),\mathcal{V}^A)_{(t,h)}$ for $t=1,\ldots,m$ and $h=1,\ldots,n_B$. For any combination of $t=1,\ldots,m$ and $h=1,\ldots,n_B$, and each $(\mathfrak{F}^{(j-1)}=(k_1,\ldots,k_m),\mathcal{V}^A)_{(t,h)}\in\mathcal{L}^{(t,h,j-1)}$, to generate a new state in $\mathcal{L}^{(t,h,j)}$, there are two choices to consider:

(1) Job $J_j{}^A$ is not completed just-in-time, i.e., assign job $J_j{}^A$ to set T. In this case, include state $(\mathfrak{F}^{(j)}=\mathfrak{F}^{(j-1)},\mathcal{V}^A)_{(t,h)}$ in $\mathcal{L}^{(t,h,j)}$;
(2) Job $J_j{}^A$ is completed just-in-time on some machine M_i, $i=1,\ldots,m$, i.e., include job $J_j{}^A$ in set E_i after job $J^A_{k_i}$. Since there is an unavailability interval $[d^B_h-p^B_{th}, d^B_h)$ on machine M_t, there are two subcases to consider:
Subcase a: If $i\ne t$ and $d^A_{k_i}+p^A_{ij}\le d^A_j$, then include state $(\mathfrak{F}^{(j)}=(k_1,\ldots,k_{i-1},j,k_{i+1},\ldots,k_m),\mathcal{V}^A+w^A_j)$ in $\mathcal{L}^{(t,h,j)}$;
Subcase b: If $i=t$ and one of the following assertions holds:
- job J^B_h is completed just-in-time prior to $J^A_{k_i}$ on machine M_i, i.e., $d^B_h+p^A_{ik_i}\le d^A_{k_i}$ and $d^A_{k_i}+p^A_{ij}\le d^A_j$,
- job $J_j{}^A$ can be completed just-in-time prior to J^B_h on machine M_i, i.e., $d^A_{k_i}+p^A_{ij}\le d^A_j$ and $d^A_j+p^B_{ih}\le d^B_h$,
- job $J_j{}^A$can be completed just-in-time immediately after J^B_h on machine M_i, implying that $J^A_{k_i}$ is a just-in-time job right prior to J^B_h, i.e., $d^A_{k_i}+p^B_{ih}\le d^B_h$ and $d^B_h+p^A_{ij}\le d^A_j$,

then include state $(\mathfrak{F}^{(j)}=(k_1,\ldots,k_{i-1},j,k_{i+1},\ldots,k_m),\mathcal{V}^A+w^A_j)$ in $\mathcal{L}^{(t,h,j)}$.

Summing up the above analysis, we formally present a solution algorithm for the $Rm||(\sum_{J^A_k\in E^A} w^A_k, \max_{J^B_k\in E^B} w^B_k)$ problem as follows:

Sum–Max–DP Algorithm SMDP

Step 1. [Pre-processing] Re-number the jobs in J^A in the EDD order.
Step 2. [Initialization] Set $\mathcal{L}^{(t,h,0)}=\{(\mathfrak{F}^{(0)}=(0,\ldots,0),0)_{(t,h)}\}$ for $t=1,\ldots,m$ and $h=1,\ldots,n_B$.
Step 3. [Generation] Generate $\mathcal{L}^{(t,h,j)}$ from $\mathcal{L}^{(t,h,j-1)}$.

```
For h=1 to n_B do
  For t=1 to m do
    For j=1 to n_A do
      Set L^(t,h,j) = ∅;
      For each (F^(j-1) = (k_1,...,k_m), V^A)_(t,h) ∈ L^(t,h,j-1) do
        /* Choice 1: Assigning job J_j^A to set T*/
        set L^(t,h,j) ← L^(t,h,j) ∪ {(F^(j) = F^(j-1), V^A)_(t,h)};
        /* Choice 2: Assigning job J_j^A to set E on some machine */
        For i=1 to m do
          If i ≠ t and d^A_{k_i} + p^A_ij ≤ d^A_j, then
        set
L^(t,h,j) ← L^(t,h,j) ∪ {(F^(j) = (k_1,...,k_{i-1},j,k_{i+1},...,k_m), V^A + w^A_j)};
          Endif
```

If $i=t$ and (($d_h^B+p_{ik_i}^A \le d_{k_i}^A$ and $d_{k_i}^A+p_{ij}^A \le d_j^A$) or ($d_{k_i}^A+p_{ij}^A \le d_j^A$ and $d_j^A+p_{ih}^B \le d_h^B$) or ($d_{k_i}^A+p_{ih}^B \le d_h^B$ and $d_h^B+p_{ij}^A \le d_j^A$)), then

set $\mathcal{L}^{(t,h,j)} \leftarrow \mathcal{L}^{(t,h,j)} \cup \{(\tilde{\aleph}^{(j)}=(k_1,\ldots,k_{i-1},j,k_{i+}\ 1,\ldots,k_m), \mathcal{V}^A+\ w_j^A)\}$;

Endif

Endfor

Endfor

Endfor

[Elimination] /* Update set $\mathcal{L}^{(t,h,j)}$ */

For any two states $(\mathcal{F}^{(t,h,j)}, \mathcal{V}_1^A)$ and $(\mathcal{F}^{(t,h,j)}, \mathcal{V}_2^A)$ with $\mathcal{V}_1^A \ge \mathcal{V}_2^A$, keep the first state in set $\mathcal{L}^{(t,h,j)}$.

Endfor

Endfor

Step 4. [Result] For every $U^B \in [0, \max_{J_k^B \in J^B} w_k^B]$, an efficient solution is given by the pair $(\mathcal{V}^{A*}, w_{h^*}^B)=(\mathcal{V}^A, w_k^B)$ with the maximum $\mathcal{V}^A$ value among all the states with $w_k^B \ge U^B$, and the optimal schedule can be determined by tracing back the solution from the end.

Theorem 3.2. *Algorithm SMDP solves the* $Rm||(\sum_{J_k^A \in E^A} w_k^A, \max_{J_k^B \in E^B} w_k^B)$ *problem in* $O(m^2 n_A^{m+1} n_B)$ *time.*

Proof. For any given t,h, and j, job J_j^A can be appended in a maximum of $m+1$ possible ways (assigning job J_j^A as the last just-in-time job to some machine under the necessity condition, say Eq. (1) or the set T) to the partial schedule represented by a state $(\mathcal{F}^{(j-1)}, \mathcal{V}^A)_{(t,h)} \in \mathcal{L}^{(t,h,j-1)}$. Thus, the algorithm generates all the possible states $(\mathcal{F}^{(j)}, \mathcal{V}^A)$ that correspond to the optimal partial schedules constructed according to Lemma 2.1. By Lemma 3.1, we keep only the dominated states in the elimination process. Thus, $\mathcal{L}^{(t,h,j)}$ always keeps all the states that may potentially be completed in an optimal schedule generated in later iterations. Therefore, after scheduling all the jobs, the optimal solution must be the one with $w_k^B \ge U^B$ and the largest $\mathcal{V}^A$ value in some set $\mathcal{L}^{(i,k,n)}$.

Now, we consider the time complexity of Algorithm *SSDP*. The pre-processing step needs a sorting operation, that requires $O(n_A \log n_A)$ time. In Step 3, for each state $(\tilde{\aleph}^{(j-1)}=(k_1,\ldots,k_m), \mathcal{V}^A)_{(t,h)}$ in $\mathcal{L}^{(t,h,j-1)}$, the upper bound on the number of values of the variables are the following: $\mathcal{F}^{(j-1)}$ is bounded by ${n_A}^m$ and $\mathcal{V}^A$ is bounded by $\sum_{J_k^A \in E^A} w_k^A$. Thus, for any combination of t and h, the total number of different states in $\mathcal{L}^{(t,h,j-1)}$ is at most ${n_A}^m$ due to the elimination rule. In each iteration j, there are at most $m+1$ new states generated from each state in $\mathcal{L}^{(t,h,j-1)}$. Thus, the number of new states generated is at most $(m+1)n_A^m$. However, because of the elimination rule, the number of new states generated in $\mathcal{L}^{(t,h,j)}$ is always at most $O(n_A^m)$ after the elimination process. Since before the elimination procedure, we have at most $O(mn_A^m)$ states in $\mathcal{L}^{(t,h,j)}$, the elimination process can be executed in $O(mn_A^m)$ time. Thus, the construction of $\mathcal{L}^{(t,h,j)}$ requires $O(mn_A^m)$ time. Note that t goes from 1 to m, h goes from 1 to n_B while j goes from 1 to n_A, so the generation process can be implemented in $O(m^2 n_A^{m+1} n_B)$ time. Step 4 requires $O(m^2 n_A^m n_B)$ time, hence the overall time complexity is indeed $O(m^2 n_A^{m+1} n_B)$. □

4. The $Rm||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem

We now turn to the $Rm||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem, in which both agents want to maximize their weighted numbers of just-in-time jobs. We first show that this problem is $\mathcal{NP}$-hard, followed by a pseudo-polynomial-time DP algorithm, establishing that it is $\mathcal{NP}$-hard in the ordinary sense, and then develop an FPTAS for finding an approximate Pareto solution.

4.1. Complexity analysis

Here we show that the recognition version of the problem even on a single machine is $\mathcal{NP}$-hard by a transformation from the $\mathcal{NP}$-complete *Even–Odd Partition* problem.

Theorem 4.1. *The recognition version of the* $1||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ *problem is* $\mathcal{NP}$*-complete.*

Proof. The proof is by reduction from the $\mathcal{NP}$-complete *Even–Odd Partition* problem, which is defined as follows:

Even–Odd Partition: Given a finite set $H=\{a_1, a_2, \ldots, a_{2h}\}$ of positive integers, where $\sum_{j=1}^{2h} a_j = M$, does there exist a partition of H into two disjoint subsets, H_1 and H_2, such that $\sum_{j \in H_1} a_j = \sum_{j \in H_2} a_j$ and such that for each j, $1 \le j \le h$, H_1 (and hence H_2) contains exactly one of $\{a_{2j-1}, a_{2j}\}$?

Given any instance of the *Even–Odd Partition* problem, we construct the following instance of the recognition version of the $1||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem:

$n_A = 2h,$

$p_j^A = M + a_j, \quad j=1,2,\ldots,2h,$

$w_j^A = M + a_j, \quad j=1,2,\ldots,2h,$

$d_j^A = M(2\lceil j/2 \rceil - 1) + a_j, \quad j=1,2,\ldots,2h,$

$U^A = (h+1/2)M,$

$n_B = 2h,$

$p_j^B = M - a_j, \quad j=1,2,\ldots,2h,$

$w_j^B = M - a_j, \quad j=1,2,\ldots,2h,$

$d_j^B = 2M\lceil j/2 \rceil, \quad j=1,2,\ldots,2h,$

$U^B = (h-1/2)M.$□

Assume first that the given instance of the *Even–Odd Partition* problem has a solution, and let us prove that there exists a feasible solution for the recognition version of the $1||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem such that $\sum_{J_k^A \in E^A} w_k^A \ge U^A$ and $\sum_{J_k^B \in E^B} w_k^B \ge U^B$. Let E^X be the set of X-jobs that corresponds to the set H_1 of the solution for the *Even–Odd Partition* instance, and let T^X be the set of all the other X-jobs, where $X \in \{A, B\}$. Consider a schedule in which the jobs in set E^X are assigned to be completed just-in-time. Then the total gain of the A-jobs for this schedule is $\sum_{J_k^A \in E^A} w_k^A = hM + \sum_{J_k^A \in E^A} a_k = (h+1/2)M = U^A$, while the total gain of the B-jobs is $\sum_{J_k^B \in E^B} w_k^B = hM - \sum_{J_k^B \in E^B} a_k = (h-1/2)M = U^B$. Therefore, the given schedule is a feasible solution for the recognition version of the $1||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem.

Conversely, we show that if there exists a schedule for the given instance of the recognition version of the $1||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem such that $\sum_{J_k^A \in E^A} w_k^A \ge U^A$ and $\sum_{J_k^B \in E^B} w_k^B \ge U^B$, then the *Even–Odd Partition* problem has a solution. For this schedule, the following assertions hold:

(1) $|E^B| = h$. Since there are exactly h different due dates of the B-jobs and two jobs cannot be completed at the same time, we have $|E^B| \le h$. Moreover, $\sum_{J_k^B \in E^B} w_k^B = \sum_{J_k^B \in E^B} (M - a_k) = M|E^B| - \sum_{J_k^B \in E^B} a_k < M|E^B|$. On the other hand, $\sum_{J_k^B \in E^B} w_k^B = M|E^B| - \sum_{J_k^B \in E^B} a_k \ge U^B = (h-1/2)M$. It follows that $|E^B| > h - 1/2$. Thus,

we get $h-1/2<|E^B|\le h$, which implies that $|E^B|=h$ since both $|E^B|$ and h are integers.

(2) $E^A=E^B$. Analogous to the proof of (1), it is easy to see that $|E^A|=h$. Thus, for each pair of A-jobs (J^A_{2j-1},J^A_{2j}) for $j=1,\ldots,h$, there is exactly one job belonging to E^A. Now, for each $J^B_j\in E^B$, assume that j is an even number (the case where "j is an odd number" can be analogously analyzed), then the starting time of J_j^B is $jM-M+a_j=(j-1)M+a_j=d^A_j$, implying that job J^A_j is completed just-in-time, i.e., $J^A_j\in E^A$. It follows from $|E^A|=|E^B|$ that $E^A=E^B$.

(3) $\sum_{J^A_k\in E^A}a_k=M/2$. It follows from $\sum_{J^A_k\in E^A}w^A_k=\sum_{J^A_k\in E^A}(M+a_k)=hM+\sum_{J^A_k\in E^A}a_k\ge U^A=(h+1/2)M$ that $\sum_{J^A_k\in E^A}a_k\ge M/2$. Similarly, $\sum_{J^B_k\in E^B}w^B_k=hM-\sum_{J^B_k\in E^B}a_k\ge U^B=(h-1/2)M$ implies that $\sum_{J^B_k\in E^B}a_k\le M/2$. Thus, $\sum_{J^A_k\in E^A}a_k=M/2$ since $E^A=E^B$.

Summing up the above analysis, there is a solution for the *Even–Odd Partition* problem.□

4.2. A pseudo-polynomial-time algorithm

In this subsection, we focus on the design of pseudo-polynomial-time DP algorithm for the $Rm||(\sum_{J^A_k\in E^A}w^A_k,\sum_{J^B_k\in E^B}w^B_k)$ problem.

The DP algorithm exploits the property stated in Lemma 2.1. Hence, in what follows, we assume that the jobs in $J^A\cup J^B$ are re-numbered from J_1 to J_n in the EDD order. We next give some definitions that facilitate the design of our DP algorithm.

Let $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B)$ be a state corresponding to a feasible partial schedule for the first j jobs $\{J_1,\ldots,J_j\}$, subject to the condition that the last just-in-time job is job J_j, in which the following information is recorded:

- $\mathcal{F}^j=(k_1,\ldots,k_l=j,\ldots,k_m)$: k_i denotes that the last just-in-time job completed on machine M_i is job J_{k_i} with $k_p\ne k_q$ if $k_p\ne 0$ and $p\ne q$ (i.e., no job can be processed on two different machines), where $k_i=0$ meaning that no job is scheduled on machine M_i in the partial schedule, and $j=k_l=\max_{i=1,\ldots,m}k_i$ indicates that the current last just-in-time job is job J_j;
- $\mathcal{V}^A$: the maximum weighted number of just-in-time A-jobs in the partial schedule;
- $\mathcal{V}^B$: the maximum weighted number of just-in-time B-jobs in the partial schedule.

The following lemma presents an elimination property, which is necessary for justifying the DP algorithm.

Lemma 4.2. *For any two states $(\mathcal{F}^j,\mathcal{V}^A_1,\mathcal{V}^B_1)$ and $(\mathcal{F}^j,\mathcal{V}^A_2,\mathcal{V}^B_2)$ with $\mathcal{V}^A_1\ge\mathcal{V}^A_2$ and $\mathcal{V}^B_1\ge\mathcal{V}^B_2$, the second state can be eliminated.*

Proof. The proof is analogous to that of Lemma 3.1.□

The DP algorithm uses Lemmas 2.1 and 4.3 to solve the problem in pseudo-polynomial time by finding the entire set of non-eliminated feasible (partial) solutions. This is achieved by dynamically updating a set $\mathcal{L}^{(j)}$ of solution states $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B)$ from $\mathcal{L}^{(0)}$, $\mathcal{L}^{(1)},\ldots,\mathcal{L}^{(j-1)}$ for $j=1,\ldots,n$. We initialize the algorithm by setting $\mathcal{L}^{(0)}=\{(\mathcal{F}^0,0,0)\}$ with $\mathcal{F}^0=(0,\ldots,0)$. For any combination of $l=0,1,\ldots,j-1$ and $i=1,\ldots,m$, and each $(\mathcal{F}^l=(k_1,\ldots,k_i,\ldots,k_m),\mathcal{V}^A,\mathcal{V}^B))\in\mathcal{L}^{(l)}$, to construct $\mathcal{L}^{(j)}$, do the following:

- If J_j is an A-job and $d_{k_i}+p_{ij}\le d_j$, then include state $(\mathcal{F}^j=(k_1,\ldots,k_{i-1},j,k_{i+1},k_m),\mathcal{V}^A+w_j,\mathcal{V}^B)$ in $\mathcal{L}^{(j)}$;
- If J_j is a B-job and $d_{k_i}+p_{ij}\le d_j$, then include state $(\mathcal{F}^j=(k_1,\ldots,k_{i-1},j,k_{i+1},k_m),\mathcal{V}^A,\mathcal{V}^B+w_j)$ in $\mathcal{L}^{(j)}$.

The condition $d_{k_i}+p_{ij}\le d_j$ is necessary for including job J_j to set E right after job J_{k_i} on machine M_i according to Eq. (1).

Summing up the above analysis, we formally present a solution algorithm for the $Rm||(\sum_{J^A_k\in E^A}w^A_k,\sum_{J^B_k\in E^B}w^B_k)$ problem as follows:

Sum–Sum–DP Algorithm SSDP

Step 1. [Pre-processing] Re-number the jobs in $J^A\cup J^B$ from J_1 to J_n in the EDD order.

Step 2. [Initialization] Set $\mathcal{L}^{(0)}=\{(\mathcal{F}^0=(0,\ldots,0),0,0)\}$.

Step 3. [Generation] Generate $\mathcal{L}^{(j)}$ from $\mathcal{L}^{(0)},\mathcal{L}^{(1)},\ldots,\mathcal{L}^{(j-1)}$.
For $j=1$ to n do
Set $\mathcal{F}^j=\varnothing$;
For $l=0$ to $j-1$ do
For each $(\mathcal{F}^l=(k_1,\ldots,k_m),\mathcal{V}^A,\mathcal{V}^B)\in\mathcal{L}^{(l)}$
For $i=1$ to m do
/* Assigning job J_j to set E right after job J_{k_i} on machine M_i*/
If J_j is an A-job and $d_{k_i}+p_{ij}\le d_j$, then set $\mathcal{L}^{(j)}\leftarrow\mathcal{L}^{(j)}\cup\{(\mathcal{F}^j=(k_1,\ldots,k_{i-1},j,k_{i+1},k_m),\mathcal{V}^A+w_j,\mathcal{V}^B)\}$;
Endif
If J_j is a B-job and $d_{k_i}+p_{ij}\le d_j$, then set $\mathcal{L}^{(j)}\leftarrow\mathcal{L}^{(j)}\cup\{(\mathcal{F}^j=(k_1,\ldots,k_{i-1},j,k_{i+1},k_m),\mathcal{V}^A,\mathcal{V}^B+w_j)\}$;
Endif
Endfor
Endfor
Endfor
[Elimination] /* Update set $\mathcal{L}^{(j)}$ */
(1) For any two states $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B_1)$ and $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B_2)$ with $\mathcal{V}^B_1\ge\mathcal{V}^B_2$, keep the first state in set $\mathcal{L}^{(j)}$;
(2) For any two states $(\mathcal{F}^j,\mathcal{V}^A_1,\mathcal{V}^B)$ and $(\mathcal{F}^j,\mathcal{V}^A_2,\mathcal{V}^B)$ with $\mathcal{V}^A_1\ge\mathcal{V}^A_2$, keep the first state in set $\mathcal{L}^{(j)}$;
Endfor

Step 4. [Result] For every $U^B\in[0,\sum_{J^B_k\in J^B}w^B_k]$, an efficient solution is given by the pair $(\mathcal{V}^{A*},\mathcal{V}^{B*})=(\mathcal{V}^A,\mathcal{V}^B)$ with the maximum $\mathcal{V}^A$ value among all the states with $\mathcal{V}^B\ge U^B$, and the optimal schedule can be determined by tracing back the solution from the end.

Theorem 4.3. *Algorithm SSDP solves the $Rm||(\sum_{J^A_k\in E^A}w^A_k,\sum_{J^B_k\in E^B}w^B_k)$ problem in $O(m^2n^{m+1}\min\{W^A,W^B\})$ time.*

Proof. For each iteration on j, job J_j can be appended in a maximum of m possible ways (assigning job J_j as the last just-in-time job to a machine under the necessity condition, say Eq. (1)) to the partial schedule represented by a state $(\mathcal{F}^l,\mathcal{V}^A,\mathcal{V}^B)\in\mathcal{L}^{(l)}$, $l=0,1,\ldots,j-1$. Thus, the algorithm generates all the possible states $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B)$ corresponding to the optimal partial schedules constructed according to Lemma 2.1. By Lemma 4.3, we keep only the dominated states in the elimination process. Thus, $\mathcal{L}^{(j)}$ always keeps all the states that may potentially be completed in an optimal schedule generated in later iterations. Therefore, after scheduling all the jobs, the optimal solution must be the one with $\mathcal{V}^B\ge U^B$ and the largest $\mathcal{V}^A$ value in some set $\mathcal{L}^{(j)}$.

Now, we consider the time complexity of Algorithm *SSDP*. The pre-processing step requires a sorting operation, which takes $O(n\log n)$ time. In Step 2, for each state $(\mathcal{F}^l,\mathcal{V}^A,\mathcal{V}^B)\in\mathcal{L}^{(j-1)}$, the upper bound on the number of values of the variables are the following: $\mathcal{F}^l$ is bounded by mn^{m-1} (there are at most m possible ways to assign job J_j as the last just-in-time job to a machine under the necessity condition, and once the assignment of job J_j is determined,

Please cite this article as: Yin Y, et al. Just-in-time scheduling with two competing agents on unrelated parallel machines. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.09.010

there are at most n^{m-1} possible vectors $(k_1,\ldots,k_{i-1},j,k_{i+1},k_m)$, and $\mathcal{V}^A$ and $\mathcal{V}^B$ have at most W^A and W^B possible values, respectively. Due to the elimination rules, $\min\{W^A,W^B\}$ is an upper bound on the number of different combinations of $\mathcal{V}^A$ and $\mathcal{V}^B$. In each iteration, there are at most m new states generated from each state in $\mathcal{L}^{(j-1)}$. Moreover, l takes from the values 0 to $j-1$. Thus, the construction of $\mathcal{L}^{(k)}$ requires $O(m^2n^m \min\{W^A,W^B\})$ time, which is also the time required for the elimination process. After n iterations, the generation process can be implemented in $O(m^2n^{m+1}\min\{W^A,W^B\})$ time. Step 4 requires $O(m^2n^m \min\{W^A,W^B\})$ time, hence the overall time complexity is indeed $O(m^2n^{m+1}\min\{W^A,W^B\})$. □

Note that Algorithm *SSDP* can also be used to solve the $Rm||(\sum_{J_k^A\in E^A}w_k^A,|E^B|)$ problem, which can be regarded as a special case of the $Rm||(\sum_{J_k^A\in E^A}w_k^A,\sum_{J_k^B\in E^B}w_k^B)$ problem, where $|E^B|$ denotes the number of just-in-time B-jobs. In fact, we only need to let $\mathcal{V}^B$ denote the number of just-in-time B-jobs in the partial schedule. Hence, we obtain the following result.

Corollary 4.4. *The* $Rm||(\sum_{J_k^A\in E^A}w_k^A,|E^B|)$ *problem can be solved in* $O(m^2n^{m+1}n_B)$ *time.*

4.3. A fully polynomial-time approximation scheme (FPTAS)

In this subsection we show how to convert *SSDP* into an FPTAS for finding an approximate Pareto solution on the trade-off curve by using the static interval partitioning approach originally proposed by Sahni [18]. Recall that, for any $0<\varepsilon<1$, an algorithm A_ε is called an *ε-approximation algorithm* for a maximization problem if we have $Z\le(1-\varepsilon)Z^*$ for all the instances, where Z denotes the value of the solution given by algorithm A_ε and Z^* is the value of the optimal solution value [18]. A family of approximation algorithms $\{A_\varepsilon\}$ defines an FPTAS if, for any $0<\varepsilon<1$, A_ε is an ε-approximation algorithm that is polynomial in n and $1/\varepsilon$.

For every $U^B\in[0,\sum_{J_k^B\in E^B}w_k^B]$ and $0<\varepsilon<1$, let $\delta=\varepsilon U^B/n$. We split the interval $[0,U^B]$ into $\lceil n/\varepsilon\rceil$ equal subintervals of size δ as follows:

$$[0,\delta),[\delta,2\delta),\ldots,\left[\left(\lceil\frac{U^B}{\delta}\rceil-2\right)\delta,\left(\lceil\frac{U^B}{\delta}\rceil-1\right)\delta\right),\left[\left(\lceil\frac{U^B}{\delta}\rceil-1\right)\delta,U^B\right].$$

This partitions $[0,\sum w_k^B]$ into a set of $\lceil n/\varepsilon\rceil+1$ subintervals.

In developing the approximation scheme, our Sum–Sum Approximation Algorithm *SSAA*(U^B) trims down every state set $\mathcal{L}^{(j)}$ in Algorithm *SSDP* to a relatively small state set $\tilde{\mathcal{L}}^{(j)}$ at the end of the jth phase, $j=1,2,\ldots,n$. The resulting state set $\tilde{\mathcal{L}}^{(j)}$ satisfies the following properties:

- $\tilde{\mathcal{L}}^{(j)}$ is a subset of $\mathcal{L}^{(j)}$;
- $\tilde{\mathcal{L}}^{(j)}$ contains at most $m(j-1)(j-2)\ldots(j-m+1)$ states whose $\mathcal{V}^B$ values fall within the same subinterval;
- For every state $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B)$ in the untrimmed state set $\mathcal{L}^{(j)}$, $\tilde{\mathcal{L}}^{(j)}$ contains some state $(\mathcal{F}^j,\tilde{\mathcal{V}}^A,\tilde{\mathcal{V}}^B)$ such that $\tilde{\mathcal{V}}^B$ and $\mathcal{V}^B$ fall within in the same subinterval with $\tilde{\mathcal{V}}^A\ge\mathcal{V}^A$.

We give a formal description of the algorithm as follows:

Sum–Sum Approximation Algorithm SSAA (U^B)

Step 1. [Pre-processing] Re-number the jobs in $J^A\cup J^B$ from J_1 to J_n in the EDD order.

Step 2. [Partitioning] Partition the interval $[0,\sum w_k^B]$ into $\lceil n/\varepsilon\rceil+1$ subintervals as follows:

$$[0,\delta),[\delta,2\delta),\ldots,\left[\left(\lceil\frac{U^B}{\delta}\rceil-2\right)\delta,\left(\lceil\frac{U^B}{\delta}\rceil-1\right)\delta\right),\left[\left(\lceil\frac{U^B}{\delta}\rceil-1\right)\delta,U^B\right],(U^B,\sum w_k^B].$$

Step 3. [Initialization] Set $\tilde{\mathcal{L}}^{(0)}=\{(\mathcal{F}^0=(0,\ldots,0),0,0)\}$.

Step 4. [Generation]: Generate $\tilde{\mathcal{L}}^{(j)}$ from $\tilde{\mathcal{L}}^{(0)},\tilde{\mathcal{L}}^{(1)},\ldots,\tilde{\mathcal{L}}^{(j-1)}$.
For $j=1$ to n do
Set $\tilde{\mathcal{L}}^{(j)}=\varnothing$;
For $l=0$ to $j-1$ do
For each $(\mathcal{F}^l=(k_1,\ldots,k_m),\mathcal{V}^A,\mathcal{V}^B)\in\tilde{\mathcal{L}}^{(l)}$
For $i=1$ to m do
/* Assigning job J_j to set E right after job J_{k_i} on machine M_i^* /
If J_j is an A-job and $d_{k_i}+p_{ij}\le d_j$, then set
$\tilde{\mathcal{L}}^{(j)}\leftarrow\tilde{\mathcal{L}}^{(j)}\cup\{(\mathcal{F}^j=(k_1,\ldots,k_{i-1},j,k_{i+1},k_m),\mathcal{V}^A+w_j,\mathcal{V}^B)\}$;
Endif
If J_j is a B-job and $d_{k_i}+p_{ij}\le d_j$, then set
$\tilde{\mathcal{L}}^{(j)}\leftarrow\tilde{\mathcal{L}}^{(j)}\cup\{(\mathcal{F}^j=(k_1,\ldots,k_{i-1},j,k_{i+1},k_m),\mathcal{V}^A,\mathcal{V}^B+w_j)\}$;
Endif
Endfor
Endfor
Endfor
[Elimination] /* Update set $\tilde{\mathcal{L}}^{(j)}$ */
For any two states $(\mathcal{F}^j,\mathcal{V}_1^A,\mathcal{V}_1^B)$ and $(\mathcal{F}^j,\mathcal{V}_2^A)$ in $\tilde{\mathcal{L}}^{(j)}$, where $\mathcal{V}_1^B$ and $\mathcal{V}_2^B$ are in the same subinterval with $\mathcal{V}_1^A\ge\mathcal{V}_2^A$, keep the first state in $\tilde{\mathcal{L}}^{(j)}$.
Endfor

Step 5. [Result]: The efficient solution is given by the pair $(\mathcal{V}^{A*},\mathcal{V}^{B*})=(\mathcal{V}^A,\mathcal{V}^B)$ with the maximum $\mathcal{V}^A$ value among all the states with $\mathcal{V}^B\ge\frac{(\lceil n/\varepsilon\rceil-1)\varepsilon U^B}{n}$, and the corresponding schedule can be determined by tracing back the solution from the end.

Lemma 4.5. *For any eliminated state* $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B)\in\mathcal{L}^{(j)}$, *there exists a state* $(\mathcal{F}^j,\tilde{\mathcal{V}}^A,\tilde{\mathcal{V}}^B)\in\tilde{\mathcal{L}}^{(j)}$ *such that* $\tilde{\mathcal{V}}^A\ge\mathcal{V}^A$, *and* $\tilde{\mathcal{V}}^B\ge\mathcal{V}^B-j\delta$ *if* $\tilde{\mathcal{V}}^B$ *falls within the same subinterval of* $[0,U^B]$ *as* $\mathcal{V}^B$ *and* $\tilde{\mathcal{V}}^B>U^B$ *if* $\tilde{\mathcal{V}}^B$ *falls within the interval* $(U^B,\sum w_k^B]$.

Proof. We prove the lemma by induction on j. According to the eliminating process of Algorithm *SSAA*(U^B), for every eliminated state $(\mathcal{F}^1,\mathcal{V}^A,\mathcal{V}^B)\in\mathcal{L}^{(1)}$, we keep an alternative state $(\mathcal{F}^1,\tilde{\mathcal{V}}^A,\tilde{\mathcal{V}}^B)$, where $\tilde{\mathcal{V}}^A\ge\mathcal{V}^A$, and $\tilde{\mathcal{V}}^B$ falls within the same subinterval of $[0,U^B]$ as $\mathcal{V}^B$ or $\tilde{\mathcal{V}}^B$ falls in the subinterval $(U^B,\sum w_k^B]$. In the former case, we have $\mathcal{V}^B-\tilde{\mathcal{V}}^B\le\delta$, and in the latter case, we have $\tilde{\mathcal{V}}^B>U^B$. Hence the result holds for $j=1$.

By the induction hypothesis, we assume that the result holds for any $l=1,\ldots,j-1$, i.e., for any eliminated state $(\mathcal{F}^l,\mathcal{V}^A,\mathcal{V}^B))\in\mathcal{L}^{(l)}$, there exists a state $(\mathcal{F}^l,\tilde{\mathcal{V}}^A,\tilde{\mathcal{V}}^B)\in\tilde{\mathcal{L}}^{(l)}$ such that $\tilde{\mathcal{V}}^A\ge\mathcal{V}^A$, and $\tilde{\mathcal{V}}^B\ge\mathcal{V}^B-l\delta$ if $\tilde{\mathcal{V}}^B$ falls within the same subinterval of $[0,U^B]$ as $\mathcal{V}^B$ and $\tilde{\mathcal{V}}^B>U^B$ if $\tilde{\mathcal{V}}^B$ falls within the interval $(U^B,\sum w_k^B]$. We now prove that the result also holds for $l=j$, i.e., for any eliminated state $(\mathcal{F}^j,\mathcal{V}^A,\mathcal{V}^B)\in\mathcal{L}^{(j)}$, there exists a state $(\mathcal{F}^j,\tilde{\mathcal{V}}^A,\tilde{\mathcal{V}}^B)\in\tilde{\mathcal{L}}^{(j)}$ such that $\tilde{\mathcal{V}}^A\ge\mathcal{V}^A$, and $\tilde{\mathcal{V}}^B\ge\mathcal{V}^B-j\delta$ if $\tilde{\mathcal{V}}^B$ falls within the same subinterval of $[0,U^B]$ as $\mathcal{V}^B$ and $\tilde{\mathcal{V}}^B>U^B$ if $\tilde{\mathcal{V}}^B$ falls within the interval $(U^B,\sum w_k^B]$.

Consider an arbitrary state $(\mathcal{F}^j, \mathcal{V}^A, \mathcal{V}^B) \in \mathcal{L}^{(j)}$. While implementing Algorithm *SSDP*, the state $(\mathcal{F}^j = (k_1, \ldots, k_{i-1}, j, k_{i+1}, \ldots, k_m), \mathcal{V}^A, \mathcal{V}^B)$ is constructed either from the state $(\mathcal{F}^s = (k_1, \ldots, k_{i-1}, h, k_{i+1}, \ldots, k_m), \mathcal{V}^A - w_j, \mathcal{V}^B)$ (case 1), or from the sate $(\mathcal{F}^s = (k_1, \ldots, k_{i-1}, h, k_{i+1}, \ldots, k_m), \mathcal{V}^A, \mathcal{V}^B - w_j)$ (case 2), where $s = \max\{h, k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_m\} < j$. Next, we only show that the result holds for case 1. The proof of case 2 is analogous.

The case where $(\mathcal{F}^j = (k_1, \ldots, k_{i-1}, j, k_{i+1}, \ldots, k_m), \mathcal{V}^A, \mathcal{V}^B)$ is constructed from the state $(\mathcal{F}^s = (k_1, \ldots, k_{i-1}, h, k_{i+1}, \ldots, k_m), \mathcal{V}^A - w_j, \mathcal{V}^B)$ implies that J_j is an A-job and $d_h + p_{ij} \le d_j$. Then, according to the induction assumption, there exists a state $(\mathcal{F}^s, \overline{\mathcal{V}}^A, \overline{\mathcal{V}}^B) \in \tilde{\mathcal{L}}^{(s)}$ such that $\overline{\mathcal{V}}^A \ge \mathcal{V}^A - w_j$ and $\overline{\mathcal{V}}^B \ge \mathcal{V}^B - s\delta$. Since J_j is an A-job and $d_h + p_{ij} \le d_j$, during the implementation of the [Generation] procedure, state $(\mathcal{F}^j, \overline{\mathcal{V}}^A + w_j, \overline{\mathcal{V}}^B)$ is constructed. It follows directly from the elimination process in $SSAA(U^B)$ that there exists a state $(\mathcal{F}^j, \tilde{\mathcal{V}}^A, \tilde{\mathcal{V}}^B)$ such that $\tilde{\mathcal{V}}^A \ge \overline{\mathcal{V}}^A + w_j \ge \mathcal{V}^A - w_j + w_j = \mathcal{V}^A$ and $\tilde{\mathcal{V}}^B \ge \overline{\mathcal{V}}^B - \delta \ge \mathcal{V}^B - (s+1)\delta \ge \mathcal{V}^B - j\delta$ if $\tilde{\mathcal{V}}^B$ falls within the same subinterval of $[0, U^B]$ as $\overline{\mathcal{V}}^B$ and $\tilde{\mathcal{V}}^B > U^B$ if $\tilde{\mathcal{V}}^B$ falls within the interval $(U^B, \sum w_k^B]$. This completes the proof.□

Theorem 4.6. *For any $0 < \varepsilon < 1$ and a Pareto-optimal solution (U^A, U^B), Algorithm $SSAA(U^B)$ finds in $O(m^2 n^{m+2}/\varepsilon)$ time a solution pair $(\tilde{\mathcal{V}}^A, \tilde{\mathcal{V}}^B)$ such that $\tilde{\mathcal{V}}^A \ge U^A$ and $\tilde{\mathcal{V}}^B \ge (1-\varepsilon)U^B$.*

Proof. Let $(\mathcal{F}^j, \mathcal{V}^A, \mathcal{V}^B)$ be a state corresponding to the Pareto-optimal solution (U^A, U^B), i.e., $\mathcal{V}^A = U^A$ and $\mathcal{V}^B = U^B$. If it has been eliminated during the implementation of Algorithm $SSAA(U^B)$, there exists a non-eliminated state $(\mathcal{F}^j, \tilde{\mathcal{V}}^A, \tilde{\mathcal{V}}^B) \in \tilde{\mathcal{L}}^{(j)}$ such that $\tilde{\mathcal{V}}^A \ge \mathcal{V}^A = U^A$, and $\tilde{\mathcal{V}}^B \ge \mathcal{V}^B - j\delta \ge \mathcal{V}^B - n\delta = U^B - \varepsilon U^B = (1-\varepsilon)U^B$ if $\tilde{\mathcal{V}}^B$ falls within the same subinterval of $[0, U^B]$ as $\mathcal{V}^B$ and $\tilde{\mathcal{V}}^B > U^B \ge (1-\varepsilon)U^B$ if $\tilde{\mathcal{V}}^B$ falls within the interval $(U^B, \sum w_k^B]$, as required.

Now we turn to the time complexity of Algorithm $SSAA(U^B)$. For each iteration in Algorithm $SSAA(U^B)$, the whole value interval $[0, \sum_{J_k^B \in E^B} w_k^B]$ is divided into $\lceil n/\varepsilon \rceil + 1$ subintervals. By the proof of Theorem 4.3, we have $|\tilde{\mathcal{L}}^{(j)}| \le m^2 n^{m-1}(\lceil n/\varepsilon \rceil + 1)$ and the time complexity of Algorithm $SSAA(U^B)$ is indeed $O(m^2 n^{m+2}/\varepsilon)$.□

5. Concluding remarks

In this paper we consider the just-in-time scheduling involving two agents that compete for the usage of m unrelated parallel machines, where one agent wants to maximize the weighted number of its just-in-time jobs, while the other agent wants to maximize its maximum gain or to maximize the weighted number of its just-in-time jobs. The goal is to find Pareto-optimal solutions for each combination of the two agents' criteria. When the number of machines is part of the problem instance, both the addressed problems are *NP*-hard in the strong sense. When the number of machines is fixed, we show that the $Rm||(\sum_{J_k^A \in E^A} w_k^A, \max_{J_k^B \in E^B} w_k^B)$ problem can be solved in $O(m^2 n_A^{m+1} n_B)$ time, while the $Rm||(\sum_{J_k^A \in E^A} w_k^A, \sum_{J_k^B \in E^B} w_k^B)$ problem is $\mathcal{NP}$-hard. For the latter problem, we present a pseudo-polynomial-time algorithm that runs in $O(m^2 n^{m+1} \min\{W^A, W^B\})$ time, implying that the problem is $\mathcal{NP}$-hard in the ordinary sense, and then convert the algorithm into an FPTAS for finding a Pareto optimal solution.

For future research, it would be interesting to find out whether or not a fast polynomial-time algorithm exists for the problems on m parallel identical machines and to consider our problems in other machine settings involving multiple agents.

Acknowledgements

We thank the Editor, an Associate Editor, and anonymous referees for their many helpful comments on earlier versions of our paper. This paper was supported in part by the National Natural Science Foundation of China (Nos. 11561036, 71501024, 71301022); and in part by the Ministry of Science Technology (MOST) of Taiwan under grant numbers NSC 102-2221-E-035-070-MY3 and MOST 103-2410-H-035-022-MY2. T. C.E. Cheng was also supported in part by The Hong Kong Polytechnic University under the Fung Yiu King - Wing Hang Bank Endowed Professorship in Business Administration.

References

[1] Agnetis A, Mirchandani P, Pacciarelli D, Pacifici A. Scheduling problems with two competing agents. Operations Research 2004;42(2):229–242.
[2] Arkin EM, Silverberg EL. Scheduling jobs with fixed start and finish times. Discrete Applied Mathematics 1987;18:1–8.
[3] Baker KR, Smith JC. A multiple-criterion model for machine scheduling. Journal of Scheduling 2003;6:7–16.
[5] Bouzina KI, Emmonss H. Interval scheduling on identical machines. Journal of Global Optimization 1996;9:379–393.
[6] Carlisle MC, Lloyd EL. On the *k*-coloring of intervals. Discrete Applied Mathematics 1995;59:225–235.
[7] Čepek O, Sung SC. A quadratic time algorithm to maximize the number of just-in-time jobs on identical parallel machines. Computers and Operations Research 2005;32:3265–3271.
[8] Choi BC, Yoon SH. Maximizing the weighted number of just-in-time jobs in flow shop scheduling. Journal of Scheduling 2007;10:237–243.
[9] Defraeye M, Nieuwenhuyse IV. Staffing and scheduling under nonstationary demand for service: a literature review. Omega 2016;58:4–25.
[10] Elalouf A, Levner E, Tang H. An improved FPTAS for maximizing the weighted number of just-in-time jobs in a two-machine flow shop problem. Journal of Scheduling 2013;16:429–435.
[11] Frank A. On chains and antichains families of a partially ordered set. Journal of Combinatorial Theory Series B 1980;29:176–184.
[12] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5:287–326.
[13] Hsiao JY, Tang CY, Chang RS. An efficient algorithm for finding a maximum weight 2-independent set of interval graphs. Information Processing Letters 1992;43:229–235.
[14] Kovalyov MY, Ng CT, Cheng TCE. Fixed interval scheduling: models, applications, computational complexity and algorithms. European Journal of Operational Research 2007;178:331–342.
[15] Lann A, Mosheiov G. Single machine scheduling to minimize the number of early and tardy jobs. Computers and Operations Research 1996;23:765–781.
[16] Leyvand Y, Shabtay D, Steiner G, Yedidsion L. Just-in-time scheduling with controllable processing times on parallel machines. Journal of Combining Optimization 2010;19:347–368.
[17] Perez-Gonzalez P, Framinan JM. A common framework and taxonomy for multicriteria scheduling problem with interfering and competing jobs: multi-agent scheduling problems. European Journal of Operational Research 2014;235:1–16.
[18] Sahni S. Algorithms for scheduling independent tasks. Journal of the ACM 1976;23(1):116–127.
[19] Shabtay D. The just-in-time scheduling problem in a flowshop scheduling system. European Journal of Operational Research 2012;216(3):521–532.
[20] Shabtay D, Bensoussan Y. Maximizing the weighted number of just-in-time jobs in several two-machine scheduling systems. Journal of Scheduling 2012;15(1):39–47.
[21] Shabtay D, Bensoussan Y, Kaspi M. A bicriteria approach to maximize the weighted number of just-in-time jobs and to minimize the total resource consumption cost in a two-machine flow-shop scheduling system. International Journal of Production Economics 2012;136:67–74.
[23] Sung SC, Vlach M. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. Journal of Scheduling 2005;8:453–460.
[24] Sterna M. A survey of scheduling problems with late work criteria. Omega 2011;39:120–129.
[25] T'kindt V, Billaut JC. Multicriteria scheduling: theory, models and algorithms. 2nd ed.. Berlin: Springer; 2006.
[26] Yannakakis M, Gavril F. The maximum *k*-colorable subgraph problem for chordal graphs. Information Processing Letters 1987;24:133–137.
[27] Yeniseya MM, Yagmahanb B. Multi-objective permutation flow shop scheduling problem: literature review, classification and current trends. Omega 2014;45:119–135.
[28] Yin Y, Cheng TCE, Hsu C-J, Wu C-C. Single-machine batch delivery scheduling with an assignable common due window. Omega 2013;41:216–225.

Please cite this article as: Yin Y, et al. Just-in-time scheduling with two competing agents on unrelated parallel machines. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.09.010