Computers & Operations Research 40 (2013) 353–361

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Two-agent singe-machine scheduling with release times to minimize the total weighted completion time

T.C.E. Cheng [a], Yu-Hsiang Chung [b], Shan-Ci Liao [c], Wen-Chiung Lee [c,*]

[a] *Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
[b] *Department of Industrial & Engineering Management, National Chiao Tung University, Hsinchu, Taiwan*
[c] *Department of Statistics, Feng Chia University, Taichung, Taiwan*

ARTICLE INFO

Available online 23 July 2012

Keywords:
Scheduling
Total weighted completion time
Maximum lateness
Two agents

ABSTRACT

In many management situations multiple agents pursuing different objectives compete on the usage of common processing resources. In this paper we study a two-agent single-machine scheduling problem with release times where the objective is to minimize the total weighted completion time of the jobs of one agent with the constraint that the maximum lateness of the jobs of the other agent does not exceed a given limit. We propose a branch-and-bound algorithm to solve the problem, and a primary and a secondary simulated annealing algorithm to find near-optimal solutions. We conduct computational experiments to test the effectiveness of the algorithms. The computational results show that the branch-and-bound algorithm can solve most of the problem instances with up to 24 jobs in a reasonable amount of time and the primary simulated annealing algorithm performs well with an average percentage error of less than 0.5% for all the tested cases.

1. Introduction

In traditional scheduling models, there is a single criterion for all the jobs. However, in the real world, customers are distinct in that they pursue different objectives. For instance, Baker and Smith [1] give an example of a prototype shop being shared by a research and development (R&D) department, which tests new designs, and by a manufacturing engineering department, which runs experiments to improve the robustness of production processes. In this context, the R&D customers might be concerned about meeting due-dates, while the engineering customers might be more concerned about quick response time. Kim et al. [2] point out that in project scheduling the major concern is with negotiation to resolve conflicts whenever the agents find their own schedules unacceptable. Schultz et al. [3] remark that in telecommunications services, the problem is to satisfy the service requirements of individual agents, who compete for the use of a commercial satellite to transfer voice, image, and text files for their clients. Agnetis et al. [4] present examples of scheduling involving multiple agents competing on the usage of common processing resources in different application environments and methodological fields, such as artificial intelligence, decision theory, and operations research.

Agnetis et al. [4] and Baker and Smith [1] first introduced the multi-agent problems into the scheduling field. They considered scheduling models with two agents and their objective functions include the total weighted completion time, the number of tardy jobs, and the maximum of regular non-decreasing functions of the job completion times. Later, Yuan et al. [5] revised the dynamic programming recursion formulae in Baker and Smith [1] and derived a polynomial-time algorithm for the same problem. Ng et al. [6] considered a single-machine two-agent problem where the objective is to minimize the total completion time of one agent, given that the number of tardy jobs of the other agent cannot exceed a certain number. They showed that the problem is NP-hard under high multiplicity encoding and can be solved in pseudo-polynomial time under binary encoding. Cheng et al. [7] studied the feasibility model involving more than two agents on a single machine where each agent's objective function is to minimize the total weighted number of tardy jobs. They showed that the general problem is strongly NP-complete. Agnetis et al. [8] studied the scheduling problems where several agents compete to perform their respective jobs on one sharing processing resource and the cost function depends on the job completion times only. They analyzed the complexity of various problems arising from different combinations of the cost functions of each agent. In particular, they investigated the problem of finding schedules whose cost for each agent does not exceed a given bound. Cheng et al. [9] examined scheduling problems on a single machine involving more than two agents where the objective functions are of the max-form. In addition, Liu and Tang [10] discussed two two-agent scheduling problems with the consideration of a simple linear deterioration effect on a single machine. Their objective functions are the maximum lateness and the total completion time of the jobs of one agent, given a bound on the maximum of a non-decreasing function of the job completion times of the other agent. Moreover, Lee

* Corresponding author.
E-mail address: wclee@fcu.edu.tw (W.-C. Lee).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.07.013

et al. [11] discussed a scheduling problem on a single machine involving more than two agents in which each agent is responsible for his own set of jobs and wishes to minimize the total weighted completion time of his own set of jobs. They reduced this NP-hard problem to a multi-objective short-path problem. They also provided an efficient approximation algorithm with a reasonably good worst-case ratio. Agnetis et al. [12] developed branch-and-bound algorithms for several single-machine scheduling problems with two competing agents. They used Lagrangian duals to derive bounds for the branch-and-bound algorithm in strongly polynomial time. Lee et al. [13] studied a two-agent scheduling problem in a two-machine permutation flowshop where the objective is to minimize the total tardiness of the jobs of one agent, given that the number of late jobs of the other agent is zero. Recently, Leung et al. [14] generalized the two-agent single-machine problems proposed by Agnetis et al. [4] to the case of multiple identical parallel machines. In addition, they also considered the situations where the jobs may have different release dates and preemptions may or may not be allowed. Lee et al. [15] studied a two-agent problem with a common linear deterioration rate for all the jobs on a single machine. They provided the optimal solution and several heuristic algorithms for the problem of minimizing the total weighted completion time of the jobs of one agent, given that no tardy jobs are allowed for the other agent. Liu et al. [16] considered a two-agent single-machine problem with linear aging or learning effects. For the objective of minimizing the total completion time of one agent, given that the maximum cost of the other agent cannot exceed an upper bound, they presented polynomial-time algorithms. Wan et al. [17] considered several two-agent problems with controllable job processing times on a single machine or two identical machines in parallel. For several problems of minimizing the objective of one agent, including the total completion time plus the compression cost, the maximum tardiness plus the compression cost, the maximum lateness plus the compression cost, and the total compression cost, subject to a given upper bound on the objective function of the other agent, they provided the NP-hard proofs for the more general problems and presented polynomial-time algorithms for several special cases. Lee et al. [18] considered a two-agent scheduling problem in a two-machine permutation flowshop. Their objective is to minimize the total completion time of the jobs of one agent with the constraint that no tardy jobs are allowed for the other agent. They developed a branch-and-bound and simulated annealing algorithm to derive the optimal and near optimal solutions, for the problem, respectively.

However, in real-life situations, customer orders do not necessarily arrive at the same time. To the best of our knowledge, the multi-agent scheduling problem with consideration of release times has hardly been studied in the literature. In this paper we study a two-agent scheduling problem on a single machine with job release times where the objective is to minimize the total weighted completion time of the jobs of one agent, subject to the maximum lateness of the jobs of the other agent does not exceed a given limit. The research results on the classical single-machine scheduling problem with release times to minimize the total weighed completion time can be found in [19–23]. The rest of this paper is organized as follows: we present the problem formulation in the next section. We develop a branch-and-bound algorithm incorporating several dominance properties and a lower bound in Section 3. We propose a primary simulated annealing algorithm in Section 4. We introduce a secondary simulated annealing algorithm and report the computational results in Section 5. Finally, we conclude the paper in the last section and suggest topics for future research.

2. Problem description

The problem under study can be described as follows: there are n jobs to be processed on a single machine. Each job belongs to either one of the two agents, namely AG_1 and AG_2. Associated with each job j, there is a processing time p_j, a weight w_j, a release time r_j, a due date d_j, and an agent code I_j, where $I_j=1$ if $j \in AG_1$ and $I_j=2$ if $j \in AG_2$. Given a schedule S, let $C_j(S)$ be the completion time of job j, $L_j(S)=C_j(S)-d_j$ be the lateness of job j, and $L^2_{\max}(S)=\max_{j\in AG_2}\{L_j(S)\}$ be the maximum lateness of the jobs of agent AG_2. The objective of the problem is to find a schedule that minimizes the total weighted completion time of the jobs of AG_1 with the restriction that the maximum lateness of the jobs of agent AG_2 does not exceed a given upper bound M. Since the objective function is a regular scheduling performance measure, we use the terms schedule and sequence interchangeably in this paper. Using the conventional three-field notation for describing scheduling problems, we denote our problem as $1|L^2_{\max}\le M, r_j|\sum_{j\in AG_1} w_jC_j$.

3. A branch-and-bound algorithm

When the number of jobs of agent AG_2 is zero, the problem under consideration reduces to the classical single-machine scheduling problem with release times to minimize the total weighted completion time, which is NP-hard [24]. So it is justified that we deploy the branch-and-bound technique to solve the problem under study. In this section we first provide several dominance properties, followed by a lower bound to speed up the search process in the branch-and-bound scheme, and finally the description of the branch-and-bound algorithm.

3.1. Dominance properties

First, we provide some results that help reduce the size of solution space of the problem or determine the optimal schedule under certain conditions. Since the correctness of both Theorems 1 and 2 are easy to establish, we omit their proofs.

Theorem 1. *If there is a job i such that* $r_i+p_i\le r_j$ *for any other job j, then there is an optimal sequence in which job i is scheduled first.*

Let S^* be a sequence in which the jobs of AG_1 are scheduled first in non-decreasing order of their release times, followed by the jobs of AG_2 in non-decreasing order of their due dates. Let $C_{[k]}(S^*)$, $r_{[k]}(S^*)$, and $d_{[k]}(S^*)$ denote the completion time, release time, and due date of the kth scheduled job under schedule S^*, respectively.

Theorem 2. *If* $C_{[k]}(S^*)\le r_{[k+1]}(S^*)$ *for the jobs of* AG_1 *and* $C_{[k]}(S^*)-d_{[k]}(S^*)\le M$ *for the jobs of* AG_2 , $k=1, \ldots, n$, *then* S^* *is an optimal schedule.*

Next, we provide several adjacent dominance properties. Suppose that S and S' are two job schedules and the difference between S and S' is a pairwise interchange of two adjacent jobs i and j. That is, $S=(\pi,i,j,\pi')$ and $S'=(\pi,j,i,\pi')$, where π and π' each denote a partial sequence. In addition, let t denote the completion time of the last job in π. Depending on whether jobs i and j are from agent AG_1 or AG_2, we consider the following four cases:

Case 1. Both jobs i and j are from agent AG_1 in sequences S and S'.

To show that S dominates S', it suffices to show that $C_j(S)-C_i(S')\le 0$ and $w_iC_i(S)+w_jC_j(S)<w_jC_j(S')+w_iC_i(S')$ in this case.

Property 1.1. *If* $\max\{t,r_i\}\le r_j$, $\max\{t,r_i\}+p_i\ge r_j$, *and* $p_i/w_i<p_j/w_j$, *then S dominates* S' .

Proof. Since $\max\{t,r_i\}\le r_j$, the completion times of jobs j and i in S' are

$$C_j(S')=r_j+p_j \tag{1}$$

and

$$C_i(S')=r_j+p_j+p_i \tag{2}$$

Similarly, the completion times of jobs i and j in S are

$$C_i(S)=\max\{t,r_i\}+p_i \le r_j+p_i \tag{3}$$

and

$$C_j(S)=\max\{C_i(S),r_j\}+p_j \le \max\{t,r_i,r_j\}+p_i+p_j=r_j+p_i+p_j \tag{4}$$

since $\max\{t,r_i\}\le r_j$. From (2) and (4), we have $C_j(S)-C_i(S')\le 0$. Since $p_i/w_i<p_j/w_j$, from (1) to (4), we have

$$w_iC_i(S)+w_jC_j(S)-w_jC_j(S')-w_iC_i(S')\le w_jp_i-w_ip_j<0$$

Thus, S dominates S'. □

Property 1.2. *If $t\ge \max\{r_i,r_j\}$ and $p_i/w_i<p_j/w_j$, then S dominates S'.*

Property 1.3. *If $\max\{t,r_i\}+p_i<r_j$, then S dominates S'.*

Case 2. Job i is from agent AG_1 and job j is from agent AG_2 in sequences S and S'.

To show that S dominates S', it suffices to show that $L_j(S)-M\le 0$ and $C_j(S)-C_i(S')<0$ in this case.

Property 2.1. *If $r_i<r_j$, $\max\{t,r_i\}+p_i\ge r_j$, and $d_j\ge \max\{t,r_i\}+p_i+p_j-M$, then S dominates S'.*

Property 2.2. *If $\max\{t,r_i\}+p_i<r_j$ and $d_j\ge r_j+p_j-M$, then S dominates S'.*

Case 3. Job j is from agent AG_1 and job i is from agent AG_2 in sequences S and S'.

To show that S dominates S', it suffices to show that $L_i(S)-M\le 0$ and $C_j(S)-C_i(S')<0$ in this case.

Property 3.1. *If $t>r_i$, $t+p_i\le r_j$, and $d_i\ge t+p_i-M$, then S dominates S'.*

Property 3.2. *If $t<r_i$, $r_i+p_i<r_j$, and $d_i>r_i+p_i-M$, then S dominates S'.*

Case 4. Both jobs i and j are from agent AG_2 in sequences S and S'.

To show that S dominates S', it suffices to show that $C_j(S)-C_i(S')<0$, $L_i(S)-M\le 0$, and $L_j(S)-M\le 0$ in this case.

Property 4.1. *If $r_i<t$, $t+p_i\le r_j$, $d_i\ge t+p_i-M$, and $d_j\ge r_j+p_j-M$, then S dominates S'.*

Property 4.2. *If $r_i>t$, $r_i+p_i\le r_j$, $d_i\ge r_i+p_i-M$, and $d_j\ge r_j+p_j-M$, then S dominates S'.*

Property 4.3. *If $r_i<t<r_j$, $t+p_i>r_j$, and $\min\{d_i,d_j-p_j\}\ge t+p_i-M$, then S dominates S'.*

We omit the proofs of Properties 1.2–4.3 since they are similar to that of Property 1.1.

To further speed up the search for the optimal solution, we provide a proposition to determine the sequence of the unscheduled jobs. Assume that π is a partial schedule in which the order of the first s jobs is determined and there are $(n-s)$ jobs yet to be scheduled. Let t be the completion time of the sth job. Among the $(n-s)$ unscheduled jobs, there are n_1 jobs of agent AG_1 and n_2 jobs of agent AG_2, where $n_1+n_2=n-s$. Without loss of generality, we assume that the job processing times, weights, and release times of the n_1 jobs of agent AG_1 are $p_1^1,p_2^1,\cdots,p_{n_1}^1$, $w_1^1,w_2^1,\cdots,w_{n_1}^1$, and $r_1^1,r_2^1,\cdots,r_{n_1}^1$, respectively, when they are arranged in the weighted shortest processing time (WSPT) order. Let (π,π_1^*,π_2^*) denote a sequence, where π_1^* consists of n_1 unscheduled jobs of agent AG_1 in the WSPT order and π_2^* consists of n_2 unscheduled jobs of agent AG_2 in the earliest due date (EDD) order.

Proposition 1. *If $t+\sum_{l=1}^{k-1}p_l^1\ge r_k^1$ for $k=1,2,\ldots,n_1$ in π_1^* and there is no tardy job in π_2^*, then sequence (π,π_1^*,π_2^*) dominates sequences of the type $(\pi,-)$.*

3.2. A lower bound

In this subsection we develop a lower bound for the branch-and-bound algorithm. Let PS be a partial sequence in which s jobs are scheduled. Among the $n-s$ jobs in the set of unscheduled jobs US, there are n_1 jobs of agent AG_1 and n_2 jobs of agent AG_2, where $n_1+n_2=n-s$. We assume that for these $n-s$ unscheduled jobs, $p_{(1)}\le p_{(2)}\le\cdots\le p_{(n-s)}$ $(r_{(1)}\le r_{(2)}\le\cdots\le r_{(n-s)})$ when they are arranged in non-decreasing order of their processing (release) times. Note that $p_{(i)}$ and $r_{(i)}$ might not be associated with the same job. Furthermore, the weights of the n_1 unscheduled jobs of agent AG_1 are denoted as $w_{(1)}\ge w_{(2)}\ge\cdots\ge w_{(n_1)}$ when they are in non-increasing order of their weights and the due dates of the n_2 jobs of agent AG_2 are denoted as $d_{(1)}\le d_{(2)}\le\cdots\le d_{(n_2)}$ when they are in non-increasing order of their due dates. The idea of developing a lower bound is that we first derive a lower bound on the completion times of the unscheduled jobs, and then we assign them to agents AG_1 and AG_2, respectively. By definition, the completion time of the $(s+1)$th job is

$$C_{[s+1]}=\max\{C_{[s]},r_{[s+1]}\}+p_{[s+1]}\ge C_{[s]}+p_{(1)}$$

By induction, the completion time of the $(s+i)$th job is

$$C_{[s+i]}\ge C_{[s]}+\sum_{l=1}^{i}p_{(l)} \tag{5}$$

On the other hand, this lower bound might not be tight if the release times are large. Thus,

$$C_{[s+1]}=\max\{C_{[s]},r_{[s+1]}\}+p_{[s+1]}\ge r_{(1)}+p_{(1)}$$

By induction, we have

$$C_{[s+i]}=\max_{1\le k\le i}\left\{r_{[k]}+\sum_{l=1}^{i-k+1}p_{[k+l]}\right\}\ge \max_{1\le k\le i}\left\{r_{(k)}+\sum_{l=1}^{i-k+1}p_{(l)}\right\} \tag{6}$$

From (5) and (6), a lower bound on the completion time of the $(s+i)$th job is

$$C_{[s+i]}\ge \max\left\{t+\sum_{l=1}^{i}p_{(l)},\max_{1\le k\le i}\left\{r_{(k)}+\sum_{l=1}^{i-k+1}p_{(l)}\right\}\right\} \tag{7}$$

for $i=1,2,\ldots,n-s$. The remaining task is to assign the estimated completion times to the jobs of agents AG_1 and AG_2. The principle is to assign the completion times to the jobs of agent AG_2 as late as possible without violating the assumption that the maximum lateness of the jobs of agent AG_2 cannot exceed an upper bound.

Proposition 2. *If there is an unscheduled job j of agent AG_2 and $\max\left\{t+\sum_{l=1}^{i}p_{(l)},\max_{1\le k\le i}\left\{r_{(k)}+\sum_{l=1}^{i-k+1}p_{(l)}\right\}\right\}-M>d_j$, then it cannot be scheduled in and after the $(s+i)$th position.*

Proof. Suppose that job j is scheduled in a position h, where $h\ge s+i$, then we have from (7) that

$$C_{[h]}-d_j\ge C_{[s+i]}-d_j\ge \max\left\{t+\sum_{l=1}^{i}p_{(l)},\max_{1\le k\le i}\left\{r_{(k)}+\sum_{l=1}^{i-k+1}p_{(l)}\right\}\right\}-d_j\ge M$$

This leads to a contradiction that the maximum lateness of the jobs of agent AG_2 cannot exceed the upper bound M and this completes the proof. □

In addition, we use the notation $C^1_{(1)} \le C^1_{(2)} \le \cdots \le C^1_{(n_1)}$ and $C^2_{(1)} \le C^2_{(2)} \le \cdots \le C^2_{(n_2)}$ to denote the estimated completion times of the jobs of agents AG_1 and AG_2 when they are arranged in non-decreasing order. By Proposition 2, we can assign the jobs in a backward manner starting with the job with the largest due date. Accordingly, we present the following Algorithm 1 to compute the lower bound:

Algorithm 1.

```
Step 1: Set $ic = n-s$, $i_1 = n_1$, $i_2 = n_2$ and
        $C_{(s+i)} = \max\left\{ t + \sum_{l=1}^{i} p_{(l)}, \max_{1 \le k \le i}\left\{ r_{(k)} + \sum_{l=1}^{i-k+1} p_{(l)} \right\} \right\}$ for $i = 1,2,\ldots,n-s$
Step 2: If $i_1 = 0$, set $C^2_{(i_2)} = C_{(s+ic)}$ and go to Step 6.
Step 3: If $i_2 = 0$, set $C^1_{(i_1)} = C_{(s+ic)}$, $i_1 = i_1 - 1$, and go to Step 5.
Step 4: If $C_{(s+ic)} \le d_{(i_2)} + M$, set $C^2_{(i_2)} = C_{(s+ic)}$ and $i_2 = i_2 - 1$. Otherwise, set $C^1_{(i_1)} = C_{(s+ic)}$ and $i_1 = i_1 - 1$.
Step 5: Set $ic = ic - 1$. If $ic = 0$, go to Step 7. Otherwise, go to Step 2.
Step 6: If $C_{(s+ic)} > d_{(i_2)} + M$, eliminate this infeasible node by setting $LB = \infty$ and go to Step 7. Otherwise, set $i_2 = i_2 - 1$ and go to Step 5.
Step 7: Output the lower bound.
```

Therefore, a lower bound on the total weighted completion time of the jobs of agent AG_1 for the partial sequence PS is $LB = \sum_{j \in AG_1} w_j C_j(PS) + \sum_{j=1}^{n_1} w_{(j)} C^1_{(j)}$.

3.3. Description of the branch-and-bound algorithm

We adopt a depth-first search to execute the branching procedure. This method has the advantage that it only requires very little storage space. Our branch-and-bound algorithm assigns the jobs in a forward manner starting with the first position. In the searching tree, the algorithm chooses a branch and systematically works down the tree until it either eliminates it by virtue of the dominance properties or the lower bound, or reaches its final node, in which case the resultant sequence either replaces the incumbent solution or is eliminated. We present an outline of the branch-and-bound algorithm as follows:

Step 1. {Initialization} Implement SA_1 (to be discussed in the next section) to obtain a sequence as the initial incumbent solution.
Step 2. {Branching & Reduction} Apply Theorem 2 to determine the optimal subsequence. Next, apply Theorem 1 and Properties 1.1–4.3 to eliminate the dominated partial sequences. For the non-dominated nodes, apply Proposition 1 to determine the sequence of the unscheduled jobs.
Step 3. {Bounding} For the non-dominated nodes, compute the lower bound on the total weighted completion time of the jobs of agent AG_1. If the lower bound on the total weighted completion time for the partial sequence is greater than the incumbent solution, eliminate that node and all the nodes beyond it in the branch. If the value of the completed sequence is less than the incumbent solution, replace it as the new solution. Otherwise, eliminate it.

Table 1
Data of the illustrative example.

i	1	2	3	4	5
AG_i	1	1	1	2	2
p_i	12	7	5	23	10
w_i			2		
d_i	–	–	–	55	60
r_i	1	22	14	30	3

In the following, we present an example to illustrate the procedures of the proposed branch-and-bound algorithm.

Example 1. There are five jobs, $M = 5$, and Table 1 lists their information, in which AG_i, p_i, w_i, d_i, and r_i denote the agent code, processing time, weight, due date, and release time of job i, respectively. The steps to execute the proposed branch-and-bound algorithm are as follows:

Step 1. We execute SA_1 to obtain the sequence (5, 1, 3, 2, 4) as the initial incumbent solution with the objective value 321.
Step 2. For the node (1, -, -, -, -), since the condition of Theorem 2 holds, we obtain the optimal subsequence (1, 3, 2, 4, 5) with the objective value 203.
Step 3. For the node (2, -, -, -, -), we delete it since its lower bound is 449, which is greater than 203. Similarly, we delete the nodes of (3, -, -, -, -) and (4, -, -, -, -) since their lower bounds are 320 and 747, respectively.
Step 4. For the node (5, -, -, -, -), since the condition of Proposition 1 holds, we obtain the optimal subsequence (5, 1, 2, 3, 4) with the objective value 320, and we delete it since it is greater than 203.

Finally, we have the optimal sequence (1, 3, 2, 4, 5) with the objective value 203.

4. A simulated annealing algorithm

The computational effort can be reduced by using a heuristic solution as an upper bound prior to applying the branch-and-bound algorithm. Furthermore, the search for the optimal solution for an instance with a large number of jobs is time consuming, but an effective heuristic can provide a time-saving approximate solution with a small margin of error. We use the simulated annealing (SA) algorithm proposed by Kirkpatrick et al. [25] to obtain a near-optimal solution in this section. The implementation of the SA algorithm consists of the following steps:

(1) **Initial sequence**: For this problem, set the initial sequence by placing the jobs of agent AG_2 in the EDD order, followed by the jobs of agent AG_1 in the WSPT order.
(2) **Neighborhood generation**: Neighborhood generation plays an important role in the efficiency of the SA method. We use three neighborhood generation methods in each iteration. They are the pairwise interchange (PI), the extraction and forward-shifted reinsertion (EFSR), and the extraction and

backward-shifted reinsertion (EBSR) movements. We choose the movement that yields the smallest value of the objective function as its neighborhood.

(3) **Acceptance probability**: Generate the probability of acceptance from the following exponential distribution

$P(accept)=\exp(-\alpha\times\Delta TC)$

where α is a control parameter and ΔTC is the change in the objective function. In addition, the method of changing α in the kth iteration is given by

$$\alpha=\frac{k}{\beta}$$

where β is an experimental factor [26]. After some pretests, we set $\beta=100{,}000$. If the total tardiness increases as a result of a random neighborhood movement, the new sequence is accepted when $P(accept)>r$, where r is a uniform random number between 0 and 1.

(4) **Stopping condition**: Our preliminary tests show that the schedule is quite stable after $200n$ iterations, where n is the number of jobs.

5. Computational experiments

We conducted computational experiments to evaluate the performance of the proposed branch-and-bound and SA algorithms. We coded all the proposed algorithms in Fortran 90 and ran them on a personal computer with an AMD Athlon(tm) 64 Processor 3500+ at a clock speed of 2.21 GHz and 1 GB RAM under Windows XP. We randomly generated the processing times from a uniform distribution over the integers 1–100. As in Chu [27], we generated the release times randomly from uniform distributions over the integers $(0,50.5n\lambda)$, where n is the number of jobs and λ is a control variable. We randomly generated the weights of the jobs of agent AG_1 from a uniform distribution over the integers between 1 and 10 and the due date of job j of agent AG_2 from a uniform distribution over the integers between $r_j+T(1-\tau-R/2)$ and $r_j+T(1-\tau+R/2)$, where R is the due date range, τ is the tardiness factor, r_j is the release time of job j, and T is the total job processing time. To ensure feasibility of each test instance, we arranged the jobs of agent AG_2 in the EDD order and re-generated the instance if the maximum lateness of the jobs of agent AG_2 exceeds the upper bound M.

The computational experiment consisted of five parts. The first part of the experiment was to test the effectiveness of the dominance properties and the lower bound, and the impact of the due date factors on the performance of the branch-and-bound and the simulated annealing algorithms. We fixed the job size at 12 and the proportion of jobs of agent AG_1 at 50%, i.e., each agent has six jobs. We set the release time factor λ at 1 and the maximum lateness at $10n$, where n is the number of jobs. We evaluated eight combinations of (τ,R), i.e., (0.25, 0.25), (0.25, 0.50), (0.25, 0.75), (0.5, 0.25), (0.5, 0.50), (0.5, 0.75), (0.75, 0.25), and (0.75, 0.50). We denote the branch-and-bound algorithm with only the dominance properties as BB_P, the branch-and-bound algorithm with the only lower bound as BB_L, the branch-and-bound algorithm with both the properties and lower bound as BB_{P+L}, and the branch-and-bound algorithm with no property and lower bound (the enumeration method) as BB_0. We recorded the mean and standard deviation of the number of nodes and the mean and standard deviation of the CPU time (in seconds) for BB_L, BB_P, and BB_{P+L}, while only the mean and the standard deviation of the CPU time (in seconds) for BB_0. For the SA algorithm SA_1, we report the mean and the standard deviation of the percentage error. The percentage error of the solution produced by SA_1 is calculated as

$(V-V^*)/V^*\times 100\%$

where V is the total weighted completion time of the solution generated by SA_1 and V^* is the total weighted completion time derived from the branch-and-bound algorithm. As a result, we randomly generated 100 replications for each condition and report the results in Table 2. It is seen that the contributions of

Table 2
Performance of the B&B and SA_1 algorithms with $n=12$, $\lambda=1.0$, $M=10n$, and $P=0.5$.

τ	R	Branch-and-bound algorithm Number of nodes						SA_1 Percentage error	
		BB_P		BB_L		BB_{P+L}			
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
0.25	0.25	4213.21	9752.70	864.440	3407.379	584.74	2178.56	0.002	0.018
	0.50	8722.94	32,236.35	635.630	1731.726	420.71	1009.04	0.000	0.000
	0.75	6034.27	15,322.25	1011.430	3444.658	653.58	2071.59	0.000	0.000
0.50	0.25	3508.56	8904.03	733.680	2494.854	442.78	1211.24	0.043	0.298
	0.50	1404.58	2379.85	372.440	769.440	257.03	454.69	0.021	0.211
	0.75	4698.59	18,470.60	514.440	1194.446	358.39	810.03	0.028	0.199
0.75	0.25	725.11	1019.37	299.890	506.679	198.75	289.87	0.009	0.080
	0.50	859.94	1092.57	250.990	298.304	172.30	179.34	0.069	0.438
		CPU times Branch-and-bound algorithm						Enumeration	
		BB_P		BB_L		BB_{P+L}			
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
0.25	0.25	0.11	0.24	0.018	0.063	0.012	0.040	230.712	13.204
	0.50	0.21	0.78	0.011	0.025	0.007	0.014	226.874	12.790
	0.75	0.15	0.35	0.018	0.054	0.011	0.029	227.895	13.350
0.50	0.25	0.08	0.16	0.013	0.043	0.008	0.023	210.039	8.889
	0.50	0.03	0.05	0.005	0.013	0.005	0.009	208.629	9.876
	0.75	0.10	0.33	0.011	0.024	0.007	0.016	210.923	10.848
0.75	0.25	0.02	0.02	0.006	0.012	0.004	0.008	195.704	5.356
	0.50	0.02	0.02	0.004	0.008	0.003	0.007	195.777	6.392

the dominance properties and the lower bound are notable in the searching process for the optimal solution in terms of execution time. The mean execution times are less than 1 second for BB_L, BB_P, and BB_{P+L}, while the mean execution time is more than 195 s for BB_0 when $n=12$. Moreover, it is noted that the lower bound is more effective than the dominance properties in terms of the number of nodes explored. Thus, we used only BB_{P+L} in the later analysis. We conducted a one-way analysis of variance (ANOVA) of the number of nodes of BB_{P+L} to test the effects of the due date factors. Table 3 reports the results. The resulting F-value for the due-date factors is 1.95 with a p-value of 0.0588, which indicates that τ and R do not have a statistically significant impact on the hardness of the problem. However, we see that the case $(\tau,R)=(0.25, 0.75)$ has the most number of nodes with an average of 653.58 among the eight cases. Thus, we used this case in the later analysis. On the other hand, as shown in Table 4, the analysis of variance of the percentage error of SA_1 has an F-value of 1.29 with a p-value of 0.25, which indicates that the due date factors have no effects on the performance of the simulated annealing algorithm.

Similar to the first part, the second part of the experiment is to test the effects of the release time control factor λ, the proportion of the jobs of agent AG_1 P, and the value of the maximum lateness M on the performance of the proposed algorithms. We fixed the job size at 12 and (τ,R) at (0.25, 0.75). We used three different values each of λ (0.2, 1.0, 3.0), P (0.25, 0.50, 0.75), and M ($10n$, $30n$, $50n$), where n is the number of jobs. As a result, we tested 27 scenarios and randomly generated 100 replications for each scenario. We present the results in Table 5. We conducted a three-way ANOVA to test the effects of the parameters on the performance of the branch-and-bound algorithm and we report the results in Table 6. The resulting F-value for the release time factor λ is 25.52 with a p-value of less than 0.001, which shows that λ has a significant impact on the performance of the branch-and-bound algorithm. A closer look at Table 5 reveals that the problems are relatively easier to solve when $\lambda=0.2$ or 3.0, and more difficult to solve when $\lambda=1.0$. The main reason for this observation is that Theorem 1 is more effective when the range of the release time is large ($\lambda=3.0$), while the other properties are more potent when the range is small ($\lambda=0.2$). The statistical test also shows that the value of the maximum lateness M is an significant factor with an F-value of 9.58 and a p-value of less than 0.001. We see from Table 6 that the problems are easier to solve when the value of M is larger. The main reason is that Theorem 2 is more effective in that case. However, the proportion of the jobs of agent AG_1 has no impact on the performance of the branch-and-bound algorithm, since the resulting F-value is 0.3 and its p-value is 0.743. As to the performance of the simulated annealing algorithm, the results in Table 7 show that none of the release time factor, the value of the maximum lateness, and the proportion of the jobs of agent AG_1 has an impact on the percentage error of SA_1. Their resulting F-values are 2.11, 1.6, and 0.47, and their associated p-values are 0.121, 0.203, and 0.628, respectively, which indicates that the performance of SA_1 is very consistent and insensitive to these parameters.

In the third part of the experiment, we examine the impact of the number of jobs. As mentioned in the first part of the experiment, we set $(\tau,R)=(0.25,0.75)$ because it had generated

Table 3
ANOVA table for the number of nodes with. $n=12$, $\lambda=1.0$, $M=10n$, and $P=0.5$.

Source	SS	DF	MS	F	p-Value
Due date factors	2.14E+07	7	3.05E+06	1.953	0.059
Error	1.24E+09	792	1.56E+06		
Total	1.26E+09	799			

Table 4
ANOVA table for the percentage error of SA_1 with. $n=12$, $\lambda=1.0$, $M=10n$, and $P=0.5$.

Source	SS	DF	MS	F	p-Value
Due date factors	0.420	7	0.060	1.293	0.251
Error	36.741	792	0.046		
Total	37.161	799			

Table 5
Performance of the B&B and SA_1 algorithms with $n=12$ and $(\tau, R)=(0.25, 0.75)$.

λ	M	P	BB_{P+L}				SA_1	
			Number of nodes		CPU time		Percentage error	
			Mean	SD	Mean	SD	Mean	SD
0.2	10n	0.25	184.0	1148.7	0.006	0.029	0.024	0.239
		0.50	218.4	902.3	0.008	0.028	0.062	0.616
		0.75	411.4	705.6	0.018	0.032	0.003	0.026
	30n	0.25	12.8	19.9	0.001	0.002	0.000	0.000
		0.50	11.1	21.8	0.001	0.002	0.000	0.000
		0.75	12.2	21.2	0.001	0.003	0.008	0.077
	50n	0.25	8.0	11.8	0.001	0.003	0.000	0.000
		0.50	11.8	14.7	0.001	0.004	0.000	0.000
		0.75	12.5	12.7	0.001	0.002	0.001	0.009
1.0	10n	0.25	1048.3	3800.4	0.013	0.046	0.000	0.000
		0.50	393.7	814.3	0.007	0.015	0.000	0.000
		0.75	228.8	403.1	0.005	0.009	0.000	0.000
	30n	0.25	119.5	525.3	0.002	0.009	0.000	0.000
		0.50	278.0	655.6	0.006	0.014	0.000	0.000
		0.75	378.3	1780.7	0.007	0.032	0.000	0.000
	50n	0.25	125.0	568.5	0.001	0.004	0.000	0.000
		0.50	578.8	1762.3	0.012	0.032	0.000	0.000
		0.75	213.7	470.2	0.004	0.008	0.000	0.000
3.0	10n	0.25	80.2	254.3	0.001	0.003	0.000	0.000
		0.50	55.1	67.3	0.001	0.004	0.000	0.000
		0.75	73.6	85.4	0.001	0.004	0.000	0.000
	30n	0.25	97.5	389.1	0.002	0.005	0.000	0.000
		0.50	63.0	94.7	0.001	0.004	0.000	0.000
		0.75	64.2	152.3	0.001	0.004	0.000	0.000
	50n	0.25	49.6	95.9	0.001	0.003	0.000	0.000
		0.50	116.9	664.2	0.002	0.009	0.000	0.000
		0.75	49.9	52.9	0.001	0.004	0.000	0.000

Table 6
ANOVA table for the number of nodes with $n=12$ and $(\tau, R)=(0.25, 0.75)$.

Source	SS	DF	MS	F	p-value
Factor (λ)	5.03E+07	2	2.51E+07	25.52	0
Factor (M)	1.89E+07	2	9.43E+06	9.58	0
Factor (P)	5.87E+05	2	2.93E+05	0.30	0.743
Error	2.65E+09	2693	9.85E+05		
Total	2.72E+09	2699			

Table 7
ANOVA table for the percentage error of SA_1 with $n=12$ and $(\tau, R)=(0.25, 0.75)$

Source	SS	DF	MS	F	p-value
Factor (λ)	0.069	2	0.035	2.11	0.121
Factor (M)	0.052	2	0.026	1.60	0.203
Factor (P)	0.015	2	0.008	0.47	0.628
Error	44.222	2693	0.016		
Total	44.359	2699			

the most number of nodes. Since the proportion of the jobs of agent AG_1 has no impact as indicated in the second part of the experiment, we fixed it at 50%. We studied three different numbers of jobs, i.e., $n=16$, 20, and 24. In addition, we considered three different values each of λ (0.2, 1.0, 3.0) and M (10n, 30n, 50n). For each condition, we randomly generated 100 replications and we report the results in Table 8. We terminate the branch-and-bound algorithm if the number of nodes explored is over 10^8, which is approximately 2.5 h in terms of execution time. We denote the instances with number of nodes over 10^8 as unsolvable instances (UI), which are also reported. For the unsolvable instances, the mean values of the gap percentages $(UB-LB)/LB \times 100\%$ are given, where UB is the solution of SA_1 and LB is the lower bound associated with the terminated node. It is seen that, since the problem is NP-hard, the number of nodes and execution time grow exponentially as the number of jobs increases. As indicated earlier, instances with $\lambda=1$ are the most difficult to solve. It is observed that the branch-and-bound algorithm could solve most of the instances with up to 24 jobs in a reasonable amount of time. Among the 900 instances with 24 jobs, only 21 are unsolvable instances, which are all associated with $\lambda=1$. In addition, the mean gap percentages are all below 5%. As to the performance of SA_1, we see that it is very accurate with a mean percentage error of less than 0.5% for all the cases. Moreover, we observe from the mean and standard deviation that SA_1 yields the optimal solutions for most of the instances when $\lambda=1$, the most difficult cases for the branch-and-bound algorithm. Thus, we recommend SA_1 in this case.

The fourth part of the computational experiments is to test the performance of SA_1 when the number of jobs is large. It is known that the branch-and-bound algorithm with a limited number of nodes explored could serve as a good heuristic in general. Thus, we propose a second simulated annealing algorithm (the secondary SA algorithm, denoted as SA_2) by using the solution obtained by the branch-and-bound algorithm with a maximum of 100n nodes as the initial sequence, where n is the number of jobs. We tested two job sizes, i.e., $n=50$ and 100. We randomly generated 100 instances for each situation and we report the results in Table 9. We recorded the mean and standard deviation of the gap percentages (GPs). The gap percentage of the solution produced by a simulated annealing algorithm is calculated as

$$(V_i-LB)/LB \times 100\%$$

for $i=1, 2$ where V_i is the value of the total weighted completion time of the jobs of the first agent generated by the ith simulated annealing algorithm and LB is the lower bound of the terminated node of the branch-and-bound algorithm. We also recorded the mean and standard deviation of the execution time. As expected, SA_2 takes more time than SA_1, especially when the number of jobs increases and the release time range is small. In addition, the performance of SA_2 is not superior to that of SA_1, which implies that the performance of SA_1 is very good.

We conducted the final part of the computational experiments to test the performance of SA_1 in treating instances with very large numbers of jobs. We used five job sizes ranging from small to very large ($n=20$, 50, 100, 1000, and 10,000). We randomly generated a set of ten instances for each situation. Table 10 presents the results. We recorded the mean and standard deviation of the gap percentages (GPs). The gap percentage of the solution produced by the simulated annealing algorithm is calculated as

$$(V-LB)/LB \times 100\%,$$

where V is the value of the total weighted completion time of the jobs of the first agent generated by SA_1 and LB is the lower bound associated with the root node of the branch-and-bound algorithm. We also recorded the mean and standard deviation of the

Table 8
Performance of the B&B and SA_1 algorithms with $(\tau, R)=(0.25, 0.75)$ and $P=0.5$.

n	λ	M	BB_{P+L}				UI	SA_1	
			Number of nodes		CPU time			Percentage error	
			Mean	SD	Mean	SD		Mean	SD
16	0.2	10n	5177.1	43,727.9	0.43	3.52	0	0.203	0.919
		30n	71.8	114.7	0.01	0.01	0	0.038	0.269
		50n	63.7	110.7	0.01	0.01	0	0.002	0.019
	1.0	10n	13,880.6	41,690.1	0.47	1.41	0	0.000	0.000
		30n	13,221.3	43,360.5	0.48	1.66	0	0.000	0.000
		50n	15,472.7	74,318.4	0.66	3.17	0	0.000	0.000
	3.0	10n	304.5	700.5	0.01	0.02	0	0.000	0.000
		30n	289.3	1032.8	0.01	0.02	0	0.000	0.000
		50n	256.7	553.0	0.01	0.01	0	0.000	0.000
20	0.2	10n	141,594.9	1,019,717.7	27.88	204.82	0	0.065	0.447
		30n	532.9	993.2	0.07	0.12	0	0.002	0.015
		50n	1009.1	3163.1	0.13	0.39	0	0.000	0.000
	1.0	10n	1,242,077.8	4,490,595.2	59.59	221.68	0	0.000	0.000
		30n	1,391,819.7	6,564,104.4	84.55	378.86	0	0.000	0.000
		50n	261,915.9	570,324.6	12.47	26.15	0	0.000	0.000
	3.0	10n	825.5	1752.5	0.02	0.04	0	0.000	0.000
		30n	539.3	1202.7	0.01	0.03	0	0.000	0.000
		50n	1103.3	2231.6	0.03	0.05	0	0.000	0.000
24	0.2	10n	424,630.7	2,008,466.5	137.22	615.32	0	0.234	1.153
		30n	125,941.6	836,859.1	17.67	115.17	0	0.117	0.452
		50n	52,668.8	320,054.0	7.99	45.88	0	0.082	0.305
	1.0	10n	5,675,136.8	14,614,339.7	460.01	1189.86	7(3.06)	0.007	0.072
		30n	3,707,708.2	10,342,309.1	262.09	746.10	6(4.55)	0.004	0.044
		50n	4,193,437.8	9,936,155.4	331.86	762.29	8(4.57)	0.000	0.000
	3.0	10n	2044.2	3989.5	0.06	0.13	0	0.000	0.000
		30n	4038.4	13,036.9	0.11	0.34	0	0.000	0.000
		50n	4603.2	20,555.4	0.13	0.58	0	0.000	0.000

Table 9
The GP of the SA_1 and SA_2 algorithms with $(\tau, R)=(0.25, 0.75)$ and $P=0.5$

n	λ	M	SA_1				SA_2			
			Gap percentage		CPU time		Gap percentage		CPU time	
			Mean	SD	Mean	SD	Mean	SD	Mean	SD
50	0.2	10n	5.487	6.71	0.091	0.01	6.129	7.063	9.111	2.277
		30n	8.878	10.48	0.099	0.009	9.696	10.534	7.665	0.88
		50n	10.088	9.815	0.1	0.01	10.929	10.157	7.667	0.94
	1.0	10n	0.306	0.644	0.094	0.009	0.324	0.658	1.568	0.481
		30n	0.267	2.921	0.104	0.011	0.315	3.055	1.544	0.416
		50n	0.308	0.621	0.107	0.01	0.38	0.716	1.528	0.424
	3.0	10n	2.654	4.557	0.085	0.009	2.655	4.556	0.396	0.075
		30n	3.249	6.145	0.093	0.008	3.261	6.141	0.404	0.059
		50n	3.292	4.642	0.097	0.009	3.298	4.643	0.409	0.074
100	0.2	10n	3.638	8.457	0.345	0.016	4.375	8.484	108.915	15.369
		30n	4.749	6.364	0.378	0.015	5.617	6.411	101.768	7.921
		50n	4.874	6.285	0.378	0.013	5.715	6.272	108.954	9.719
	1.0	10n	0.006	1.402	0.358	0.017	0.007	1.395	15.819	4.273
		30n	0.025	0.708	0.395	0.015	0.031	1.2	15.781	4.105
		50n	0.026	1.108	0.413	0.014	0.033	1.092	15.27	4.045
	3.0	10n	0.847	0.959	0.323	0.014	0.849	0.961	2.017	0.3
		30n	0.831	1.229	0.347	0.015	0.835	1.229	2.073	0.283
		50n	0.746	1.088	0.364	0.014	0.747	1.088	2.129	0.315

Table 10
The GP of SA_1 with $(\tau, R)=(0.25, 0.75)$ and $P=0.5$.

n	Gap percentage			CPU time		
	Mean	Max	SD	Mean	Max	SD
20	189.64	393.46	80.76	0.01	0.02	0.01
50	201.72	347.92	61.03	0.04	0.06	0.01
100	193.51	386.18	47.66	0.17	0.20	0.01
1000	197.70	268.15	25.29	20.79	23.84	1.48
10,000	201.41	243.49	24.95	1687.54	1954.23	121.03

execution time. It is seen that the mean *GP* remains around 200% as the number of jobs increases, which implies that the performance of SA_1 is consistent.

6. Conclusions

In this paper we study a two-agent single-machine scheduling problem with release times. The objective is to minimize the total weighted completion time of the jobs of one agent, given that the maximum lateness of the jobs of the other agent cannot exceed a given upper bound. We propose a branch-and-bound algorithm to solve the problem, and a primary and a secondary simulated annealing algorithm to find near-optimal solutions. We conduct computational experiments to evaluate the performance of the proposed algorithms. The computational results show that the branch-and-bound algorithm can solve most of the problem instances with up to 24 jobs in a reasonable amount of time. They also show that the performance of the primary SA algorithm is very good, yielding an average percentage error of less than 0.5% for all the tested cases, and the performance is stable with respect to the due dates, proportions of jobs of the two agents, release times, and bound on the of maximum lateness of the jobs of the other agent.

Acknowledgments

We thank two anonymous referees for their helpful comments on an earlier version of our paper.

References

[1] Baker KR, Smith JC. A multiple-criterion model for machine scheduling. Journal of Scheduling 2003;6:7–16.
[2] Kim K, Paulson BC, Petrie CJ, Lesser VR. Compensatory negotiation for agent-based schedule coordination. CIFE working paper #55. Stanford, CA: Stanford University; 1999.
[3] Schultz D, OH SH, Grecas CF, Albani M, Sanchez J, Arbib C. A QoS concept for packet oriented SUMTS services. In: Proceedings of the 1st mobile summit. Thessaloniki, Greece; 2002.
[4] Agnetis A, Mirchandani PB, Pacciarelli D, Pacifici A. Scheduling problems with two competing agents. Operations Research 2004;52:229–42.
[5] Yuan JJ, Shang WP, Feng Q. A note on the scheduling with two families of jobs. Journal of Scheduling 2005;8:537–42.
[6] Ng CT, Cheng TCE, Yuan JJ. A note on the complexity of the problem of two-agent scheduling on a single machine. Journal of Combinatorial Optimization 2006;12:387–94.
[7] Cheng TCE, Ng CT, Yuan JJ. Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. Theoretical Computer Science 2006;362:273–81.
[8] Agnetis A, Pacciarelli D, Pacifici A. Multi-agent single machine scheduling. Annals of Operations Research 2007;150:3–15.
[9] Cheng TCE, Ng CT, Yuan JJ. Multi-agent scheduling on a single machine with max-form criteria. European Journal of Operational Research 2008;188: 603–9.
[10] Liu P, Tang L. Two-agent scheduling with linear deteriorating jobs on a single machine. Lecture Notes in Computer Science 2008;5092:642–50.
[11] Lee KB, Choi BC, Leung JYT, Pinedo ML. Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. Information Processing Letters 2009;109:913–7.
[12] Agnetis A, Pascale G, Pacciarelli DA. Lagrangian approach to single-machine scheduling problems with two competing agents. Journal of Scheduling 2009;12:401–15.
[13] Lee WC, Chen SK, Wu CC. Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem. Expert Systems with Applications 2010;37:6594–601.
[14] Leung JYT, Pinedo M, Wan GH. Competitive two agents scheduling and its applications. Operations Research 2010;58:458–69.
[15] Lee WC, Wang WJ, Shiau YR, Wu CC. A single-machine scheduling problem with two-agent and deteriorating jobs. Applied Mathematical Modelling 2010;34:3098–107.
[16] Liu P, Zhou X, Tang L. Two-agent single-machine scheduling with position-dependent processing times. International Journal of Advanced Manufacturing Technology 2010;48:325–31.
[17] Wan G, Vakati SR, Leung JYT, Pinedo M. Scheduling two agents with controllable processing times. European Journal of Operational Research 2010;205:528–39.
[18] Lee WC, Chen SK, Chen CW, Wu CC. A two-machine flowshop problem with two agents. Computers and Operations Research 2011;38:98–104.
[19] Hariri AMA, Potts CN. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. Discrete Applied Mathematics 1983;5:99–109.

[20] Belouadah H, Posner ME, Potts CN. Scheduling with release dates on a single machine to minimize total weighted completion time. Discrete Applied Mathematics 1992;36:213–31.

[21] Janiak A. Single machine sequencing with linear models of release dates. Naval Research Logistics 1998;45:99–113.

[22] Eren T. Minimizing the total weighted completion time on a single machine scheduling with release dates and a learning effect. Applied Mathematics and Computation 2009;208:355–8.

[23] Lee WC, Wu CC, Hsu PH. A single-machine learning effect scheduling problem with release times. Omega—The International Journal of Management Science 2010;38:3–11.

[24] Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.

[25] Kirkpatrick S, Gellat CD, Vecchi MP. Optimization by simulated annealing algorithm. Science 1983;220:671–80.

[26] Ben-Arieh D, Maimon O. Annealing method for PCB assembly scheduling on two sequential machines. International Journal of Computer Integrated Manufacturing 1992;5:361–7.

[27] Chu CB. A branch-and-bound algorithm to minimize total flow time with unequal release dates. Naval Research Logistics 1992;39:859–75.