We consider a single-machine two-agent problem where the objective is to minimize a weighted combination of the total completion time and the total tardiness of jobs from the first agent given that no tardy jobs are allowed for the second agent. A branch-and-bound algorithm is developed to derive the optimal sequence and two simulated annealing heuristic algorithms are proposed to search for the near-optimal solutions. Computational experiments are also conducted to evaluate the proposed branch-and-bound and simulated annealing algorithms.

1. Introduction

In traditional scheduling, there is a common goal to minimize for all the jobs [1–4]. In contrast to that, there is a growing interest in multiagent scheduling problems where jobs are from several customers who have different goals to pursue. For instance, Peha [5] gave the telecommunication service examples where various types of packets and service compete for the use of a commercial satellite, and the problem is to satisfy the service requirements of individual agents to transfer voice, image, and text files for their clients. Brewer and Plott [6] gave a transportation example where the agents own transportation resources and compete for the usage of the infrastructures. Kubiž and Strusevich [7] presented an example that maintenance operations complete with the real jobs for machine occupancy on maintenance planning. Baker and Smith [8] gave a sharing example of a prototype shop in which the department of research and development might be concerned about quick response time, while the department of manufacturing might be more concerned about meeting the due dates.

Agnetis et al. [9] and Baker and Smith [8] were the pioneers that brought the multiagent problems into scheduling field. Agnetis et al. [9] considered the maximum of regular functions, number of late jobs, and total weighted completion times. They obtained different scenarios depending on the objective function of each agent and on the structure of the processing system. For each scenario, they addressed the complexity of various problems. Baker and Smith [8] examined the implications of minimizing an aggregate scheduling objective function in which jobs belonging to different customers are evaluated based on their individual criteria. They demonstrated that the problem to minimize a mix of makespan, maximum lateness, or total weighted completion time is NP-hard. Cheng et al. [10] considered the feasibility model of multiagent scheduling 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 and developed the complexity results for some special cases. Ng et al. [11] addressed a single-machine two-agent problem to minimize the total completion time of the first agent given that the number of tardy jobs of the second agent cannot exceed a certain number. They showed that the problem is NP-hard under high multiplicity encoding and can be solved in pseudopolynomial time under binary encoding. Lee et al. [12] discussed a single-machine multiagent scheduling problem 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 multiobjective short-path problem. They also provided an efficient approximation algorithm with a reasonably good worst-case ratio. Agnetis et al. [13] developed the branch-and-bound algorithms for several single-machine two-agent scheduling problems. They used Lagrangian dual to derive the bounds for the branch-and-bound algorithm in strongly polynomial time. Lee et al. [14] studied a single-machine two-agent problem with deteriorating jobs in which the objective function is to minimize the total completion time of jobs from one agent given that no tardy jobs are allowed for the other agent. They provided a branch-and-bound algorithm to derive the optimal solution and several heuristic algorithms for the proposed problem. Recently, Leung et al. [15] generalized the single-machine problems proposed by Agnetis et al. [9] 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. Nong et al. [16] studied a single-machine two-agent scheduling problem for minimizing the total cost in which the cost of the first agent is the maximum weighted completion time while that of the second agent is the total weighted completion time. Liu et al. [17] considered single-machine two-agent problems with increasing linear deterioration consideration. They developed the optimal solutions for some problems where the goal is to minimize the objective function of the first agent given that the objective function of the second agent cannot exceed a certain upper bound. Cheng et al. [18] studied 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. They proposed a branch-and-bound algorithm to solve problems with up to 24 jobs. Yu et al. [19] developed the optimal solutions for several single-machine scheduling problems with two competing agents and maintenance activity. Liu et al. [20] considered single-machine scheduling with two-agent and sum-of-processing-times-based deterioration effect. They developed some polynomial-time algorithms for the respective problems.

Pinedo [1] pointed out that a customer is often concerned with multiple objectives. For instance, he might want to have a lower cost and the on-time delivery. To the best of our knowledge, multijagent scheduling problems with the consideration of multiple objectives from the same agent have seldom been discussed in the literature. In this paper, we study a two-agent scheduling problem on a single machine where the objective is to minimize the weighted combination of the total completion time and the total tardiness of jobs from the first agent given that the number of tardy jobs from the second agent is zero. The rest of the paper is organized as follows. In the next section the formulation of our problem is described. In Section 3, a branch-and-bound algorithm incorporating several elimination rules and a lower bound is constructed to speed up the search for the optimal solution. In Section 4, two simulated annealing algorithms are proposed to solve this problem. In Section 5, a computational experiment is conducted to evaluate the efficiency of the branch-and-bound algorithm and the performance of the proposed simulated annealing algorithms. A conclusion is given in the final section.

2. Problem Description

The problem we study is described as follows. There are \( n \) jobs ready to be processed. Each job belongs to either one of the two agents \( AG_1 \) or \( AG_2 \). For each job \( j \), there is a processing time \( p_j \), a due date \( d_j \), and an agent code \( \alpha_j \), where \( \alpha_j = 1 \) if job \( j \) \( \in AG_1 \) or \( \alpha_j = 2 \) if job \( j \) \( \in AG_2 \). Under a schedule \( S \), let \( C_j(S) \) be the completion time of job \( j \), let \( T_j(S) = \max\{0, C_j(S) - d_j\} \) be the tardiness of job \( j \), and \( U_j(S) = 1 \) if \( T_j(S) > 0 \) and zero otherwise. In this paper, we study a single-machine problem to minimize the weighted combination of the total completion time and the total tardiness of jobs from \( AG_1 \) given that no tardy jobs from \( AG_2 \) are allowed. Using the conventional three fields of notation, this problem is denoted by \( 1| \sum j \in AG_2 U_j = 0| \sum j \in AG_1 \theta T_j + (1 - \theta)C_j \) where \( 0 \leq \theta \leq 1 \).

3. A Branch-and-Bound Algorithm

If there is no job from agent \( AG_2 \) and \( \theta = 1 \), the problem under consideration reduces to the classical single-machine total tardiness time problem which is proved to be NP-hard by Du and Leung [21]. Therefore, a branch-and-bound algorithm might be a good way to derive the optimal solution. In this section, we will provide several dominance properties to speed up the search process.

3.1. Dominance Properties. In this subsection, we develop a nonadjacent and several adjacent dominance properties to reduce the searching scope.

Property 1. If jobs \( i, j \in AG_1, d_i \leq d_j \), and \( p_i < p_j \), then job \( i \) must precede job \( j \) in an optimal schedule.

The following propositions are the ordering criteria on a pair of adjacent jobs. Suppose that \( S \) and \( S' \) are two schedules of jobs and the only difference between them 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 \( \pi \) and \( \pi' \) each denote a partial sequence. In addition, let \( t \) denote the completion time of the last job in \( \pi \). The completion times of jobs \( i \) and \( j \) in \( S \) are

\[
C_i(S) = t + p_i, \quad (1)
\]

\[
C_j(S) = t + p_i + p_j \quad (2)
\]

Similarly, the completion times of jobs \( j \) and \( i \) in \( S' \) are

\[
C_j(S') = t + p_j, \quad (3)
\]

\[
C_i(S') = t + p_j + p_i, \quad (4)
\]
**Proposition 1.** If jobs $i, j \in AG_1$, $p_i < p_j$ and $d_i < d_j < t + p_j$, then $S$ dominates $S'$.

*Proof.* Since jobs in partial sequence $\pi$ are processed in the same order in both $S$ and $S'$ and from (2) and (4), we have

$$C_k(S) = C_k(S')$$

if job $k \in \pi$ or $\pi'$. (5)

To show that $S$ dominates $S'$, it suffices to show

$$T_j(S) = \max\left\{0, t + p_i - d_i\right\},$$
$$T_j(S) = t + p_i + p_j - d_j,$$
$$T_j(S') = t + p_j - d_i,$$
$$T_j(S') = t + p_j + p_i - d_i.$$ (7)

Suppose that $T_j(S)$ is not zero. Note that this is the more restrictive case since it comprises the case that $T_j(S)$ is zero. From $p_i < p_j$, we have

$$\theta \left[ T_j(S) + T_j(S) \right] + (1 - \theta) \left[ C_i(S) + C_j(S) \right]$$

$$= \theta \left[ T_j(S') + T_j(S') \right] + (1 - \theta) \left[ C_i(S') + C_j(S') \right]$$

$$= p_i - p_j < 0.$$ (8)

Thus, $S$ dominates $S'$.

---

**Proposition 2.** If jobs $i, j \in AG_1$, $(p_i - p_j) + \theta (d_i - t - p_j) < 0$, $d_i < t + p_i < d_j < t + p_i + p_j$, and $p_i < p_j$, then $S$ dominates $S'$.

**Proposition 3.** If jobs $i, j \in AG_1$, $d_j < t + p_i + p_j < d_i$, and $p_i - (1 - \theta) p_j < 0$, then $S$ dominates $S'$.

**Proposition 4.** If jobs $i, j \in AG_1$, $t + p_i < d_i$, $t + p_j < d_j$, $t + p_i + p_j < \max(d_i, d_j)$, and $\theta (d_i - d_j) + (1 - \theta)(p_i - p_j) < 0$, then $S$ dominates $S'$.

**Proposition 5.** If jobs $i, j \in AG_1$, $t + p_j < d_i < t + p_i + p_j$, and $\theta (t + p_j - d_j) + (p_i - p_j) < 0$, then $S$ dominates $S'$.

**Proposition 6.** If jobs $i, j \in AG_1$, $p_i < p_j$, and $t + p_i + p_j < d_j < d_i$, then $S$ dominates $S'$.

**Proposition 7.** If jobs $i, j \in AG_1$, $t + p_j < d_i < t + p_i + p_j$, $(p_i - p_j) + \theta (t + p_j - d_j) < 0$, and $d_j < t + p_i$, then $S$ dominates $S'$.
bounds for the completion times of jobs from agent AG₁. The steps are given as follows.

**Phase I**

- **Step 1.** Set \( i = n_2, s_t = \infty \), and \( t_{n_i+1} = \infty \).
- **Step 2.** If \( d^2_{i(j)} < s_t \), set \( t_i = d^2_{i(j)} - p^2_{i(j)} \). Otherwise, set \( t_i = s_t - p^2_{i(j)} \).
- **Step 3.** Set \( s_t = t_i \) and \( i = i - 1 \). If \( i \geq 1 \), go to **Step 2**.
- **Step 4.** Output \((t_1, t_2, \ldots, t_{n_i}, t_{n_i+1})\).

**Phase II**

- **Step 1.** Set \( i = 1 \) and \( k = 0 \).
- **Step 2.** Set \( k = k + 1 \).
- **Step 3.** If \( t + p^1_{i(j)} > t_k \), set \( p^1_{i(j)} = p^1_{i(j)} - (t_k - t) \), \( t = t + p^2_{i(k)} \), and go to **Step 2**. Otherwise, set \( t = t + p^1_{i(j)} \) and \( C^1_{i(j)} = 1 \).
- **Step 4.** If \( i < n_i \), set \( i = i + 1 \) and go to **Step 3**.
- **Step 5.** Output \((C^1_{(1)}, C^1_{(2)}, \ldots, C^1_{(n_i)})\).

Thus, the lower bound of the weighted combination of the total completion time and the total tardiness for PS is

\[
LB(PS) = \sum_{j \in AG \cap PS} \left( (1 - \theta) C^1_{(j)} (PS) + \theta T^1_{(j)} (PS) \right) + \sum_{j=1}^{n_i} (1 - \theta) C^1_{(j)} + \theta \max \left\{ 0, C^1_{(j)} - d^1_{(j)} \right\} .
\]  

**4. The Simulated Annealing Algorithm**

Metaheuristic algorithms have been successfully applied to solve many scheduling or optimization problems [22–25]. The simulated annealing (SA) algorithm is one of the most popular ones. It is originally proposed by Kirkpatrick et al. [26] and has been successfully applied to solve many combinatorial optimization problems. The advantage of SA algorithm is that it could avoid getting trapped in a local optimum. In this paper, we utilize the SA algorithm to derive the near-optimal solution for the proposed problem. A brief description of the SA procedure is as follows. Given an initial sequence, a new sequence is created by a random neighborhood generation. The new sequence is accepted if it has a smaller objective function than the original sequence; otherwise, it is accepted with some probability that decreases as the process evolves. The exchange condition is initially set to a high level so that a neighborhood exchange could happen frequently in early iterations. It is gradually lowered using a predetermined cooling strategy so that it becomes more difficult to exchange in later iterations unless a better solution is obtained.

The key elements of the SA algorithm are as follows.

1. **Initial Sequence.** Since no tardy job is allowed for agent AG₂ and the SPT rule yields the optimal solution for the total completion time problem, the initial sequence is constructed as follows. Jobs from agent AG₂ are first placed according to the EDD rule, followed by jobs from agent AG₁ according to the SPT rule.

2. **Neighborhood Generation.** Neighborhood generation plays an important role in the efficiency of the SA method. Three neighborhood generation methods are used in the preliminary study. They are the pairwise interchange (PI), the extraction and forward-shifted reinsertion (EFSR), and the extraction and backward-shifted reinsertion (EBSR) movements. It is observed that the PI movement yields a better solution in general. Thus, it is used in later analysis.

3. **Acceptance Probability.** In SA, solutions are accepted according to the magnitude of increase in the objective function and the temperature. The probability of acceptance is generated from an exponential distribution:

\[
P(\text{accept}) = \exp(-\alpha \times \Delta TC),
\]

where \( \alpha \) is the control parameter and \( \Delta TC \) is the change in the objective function. In addition, the method of changing \( \alpha \) at the \( k \)th iteration is obtained from Ben-Arieh and Maimon [27] and is given by

\[
\alpha = \frac{k}{\beta},
\]

where \( \beta \) is an experimental factor. After some pretests, we chose \( \beta = 6000 \). If the weighted combination of the total completion time and the total tardiness increases as a result of a random neighborhood movement, the new sequence is accepted when \( P(\text{accept}) > r \), where \( r \) is a uniform random number between 0 and 1.

4. **Objective Function.** The objective function is usually chosen to be the one that we want to minimize. Thus, in the first simulated annealing algorithm \( SA_1 \), the objective function is \( \sum_{j \in AG} \theta T^1_{(j)} + (1 - \theta)C^1_{(j)} \) and regenerates the neighborhood if it is infeasible. Since our problem is to minimize an objective function under a constraint, it is reasonable to add the constraint into the objective function. Thus, in the second simulated annealing algorithm \( SA_2 \), the objective function is modified to \( \sum_{j \in AG} \theta T^1_{(j)} + (1 - \theta)C^1_{(j)} + \lambda \sum_{j \in AG} T^2_{(j)} \) where \( \lambda \) is chosen to be 1000 after some preliminary tests.

5. **Stopping Condition.** Our preliminary tests showed that the schedule is quite stable after 400n iterations, where \( n \) is the number of jobs. Thus, 400n was used as the number of iterations.

**5. Computational Experiments**

In this section, we conducted the computational experiments to evaluate the performance of the branch-and-bound and the proposed SA algorithms. The algorithms were coded in Fortran 90 and run on a personal computer with Intel(R) Core(TM)2 Duo CPU T7500 2.20 GHz and 2.99 GB RAM under Windows XP. The job processing times were generated
Table 1: The performance of the B&B with \( n = 12 \) and \( \theta = 0.5 \).

<table>
<thead>
<tr>
<th>( P )</th>
<th>( \tau )</th>
<th>( R )</th>
<th>Number of nodes</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>275.81</td>
<td>182.83</td>
</tr>
<tr>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>389.64</td>
<td>327.66</td>
</tr>
<tr>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>496.82</td>
<td>774.90</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>378.42</td>
<td>363.44</td>
</tr>
<tr>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>504.80</td>
<td>636.00</td>
</tr>
<tr>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>473.48</td>
<td>474.64</td>
</tr>
</tbody>
</table>

Table 2: ANOVA table for the number of nodes with \( n = 12 \) and \( \theta = 0.5 \).

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>( F )</th>
<th>( P ) value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
<td>2</td>
<td>1</td>
<td>36236586</td>
<td>18118293</td>
<td>189.52</td>
</tr>
<tr>
<td>( (\tau, R) )</td>
<td>7</td>
<td>1</td>
<td>19531459</td>
<td>2790208</td>
<td>72.19</td>
</tr>
<tr>
<td>Error</td>
<td>2390</td>
<td>2</td>
<td>2.28E + 08</td>
<td>95598</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2399</td>
<td>2</td>
<td>2.28E + 08</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: The performance of the B&B with \( n = 12 \) and \( P = 0.5 \).

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \tau )</th>
<th>( R )</th>
<th>Number of nodes</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>246.56</td>
<td>174.35</td>
</tr>
<tr>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>367.78</td>
<td>604.59</td>
</tr>
<tr>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>359.06</td>
<td>396.26</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>93.20</td>
<td>61.04</td>
</tr>
<tr>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>226.18</td>
<td>264.67</td>
</tr>
<tr>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>285.38</td>
<td>297.02</td>
</tr>
</tbody>
</table>

Table 4: ANOVA table for the number of nodes with \( n = 12 \) and \( P = 0.5 \).

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DF</th>
<th>MS</th>
<th>( F )</th>
<th>( P ) value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta )</td>
<td>2</td>
<td>1</td>
<td>113054</td>
<td>56527</td>
<td>0.74</td>
</tr>
<tr>
<td>( (\tau, R) )</td>
<td>7</td>
<td>1</td>
<td>38742472</td>
<td>5534639</td>
<td>72.26</td>
</tr>
<tr>
<td>Error</td>
<td>2390</td>
<td>2</td>
<td>1.83E + 08</td>
<td>76593</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2399</td>
<td>2</td>
<td>2.22E + 08</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

from a uniform distribution over the integers 1 to 100 and the due dates of jobs were generated from a uniform distribution over the integers between \( T(1 - \tau - R/2) \) and \( T(1 - \tau + R/2) \), where \( R \) is the due date range factor, \( \tau \) is the tardiness factor, and \( T \) is the total processing time of all the jobs.

The computational experiments were divided into four parts. In the first part of the experiment, we studied the effects of the due date factors \( \tau \) and \( R \) and the proportion of jobs from the second agent \( P \) to the performance of the branch-and-bound algorithm. The job size \( n \) was 12, and the coefficient of the weighted combination of the total completion time and the total tardiness \( \theta \) was 0.5. Three values of the proportion of jobs from agent AG1 were considered, that is, \( P = 0.25, 0.5, \) and 0.75. Eight combinations of \( (\tau, R) \) values were tested, that is, \( (0.25, 0.25), (0.25, 0.50), (0.25, 0.75), (0.5, 0.25), (0.5, 0.50), (0.5, 0.75), (0.75, 0.25), \) and \( (0.75, 0.50) \). The mean and the standard deviation of the number of nodes and the mean and the standard deviation of the CPU time (in seconds) were reported for the branch-and-bound algorithm. 100 replications were randomly generated for each case and the results were presented in Table 1. A two-way analysis of variance (ANOVA) on the number of nodes of the branch-and-bound algorithm was constructed and given in Table 2. The resulting \( F \) value of factor \( P \) was 189.52 with a \( P \) value close to 0, which indicated that the values of \( P \) really affect the performance of the branch-and-bound algorithm. In fact, problems are easier to solve when the value of \( P \) is larger. The main reason is that Property 2 is more powerful in this case. In addition, the resulting \( F \) value of due date factor \( \tau \) or \( R \) was 29.19 with a \( P \) value close to 0, which indicated that
the values of $\tau$ or $R$ have statistically significant effects on the difficulty of the problem. A closer look at Table 1 revealed that problems are easier to solve as $R$ decreases or when $\tau = 0.75$. The main reason is that Property 2 and the lower bound are more powerful in those cases.

Similar to the first part of the experiment, the second part was to test the effects of the coefficient $\theta$ as well as the effects of the due date factors $\tau$ and $R$. The job size was fixed at 12, and $P$ was 0.5. Three different values of $\theta$ (0.25, 0.5, and 0.75) and eight values of $(\tau, R)$ values were tested, that is, (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). As a consequence, 24 cases were tested and 100 replications were randomly generated for each case. The results were presented in Table 3. A two-way ANOVA on the number of nodes was utilized to test the effects of the parameters on the performance of the branch-and-bound algorithm, and the result was reported in Table 4. The resulting $F$ value for the coefficient $\theta$ was 0.74 with a $P$
The value of 0.48, which implied that $\theta$ does not affect the performance of the branch-and-bound algorithm. On the other hand, the statistical test showed that the due date factors have significant effects on the performance of the branch-and-bound algorithm, and the problems are easier to solve when $\tau = 0.75$, which is consistent with the findings in the first part of the experiment.

The main purpose of the third part of the experiment was to study the impact of the number of jobs to the performance of the branch-and-bound algorithms and the accuracy of the proposed simulated annealing algorithms. Three different job sizes ($n = 16$, $20$, and $24$) were tested. Since the problems are relatively easier to solve when $\tau = 0.75$ and $\theta$ was found to be the insignificant factors in the second part of the experiments, we ignored the cases when $\tau = 0.75$ and fixed $\theta$ at 0.5 in the third part of the experiment. Two values of $\tau$ (0.25, 0.5) and three values of $P$ (0.25, 0.50, and 0.75) and of $R$ (0.25, 0.50, and 0.75) were tested. The mean and standard deviation of the number of nodes and the mean and standard deviation of the CPU time (in seconds) for the branch-and-bound algorithm were given, while only the mean and standard deviation of the error percentages of the SA algorithms were reported. The error percentage of the solution produced by SA is calculated as

$$\frac{(V - V^*)}{V^*} \times 100\%,$$  

(12)

where $V$ is the weighted combination of the total completion time and the total tardiness of the solution generated by SA and $V^*$ is the value of the optimal solution obtained from the branch-and-bound algorithm. The execution time of SA algorithms was not recorded since they were finished within a second. For each condition, 100 replications were generated and the results were given in Table 5. It is observed that the branch-and-bound algorithm could solve problems of up to 24 jobs in a reasonable amount of time. However, the number of nodes and execution time grow exponentially as the number of jobs increases since the problem is NP-hard. The most time consuming case $(n, P, \tau, R) = (24, 0.25, 0.5, 0.75)$ took an average execution time of 47.37 seconds. As to the performance of the SA algorithms, the results showed that both of them are quite accurate with a mean percentage error of less than 1% for all the tested cases. Moreover, it seemed that SA$_2$ performed better than SA$_1$ when the number of jobs is 16; however, the trend is not obvious as the number of jobs becomes larger. Thus, we would study the performance of the algorithms for large job-size problems in the last part of the experiments.

We tested two job sizes, that is, $n = 100$ and 200, in the last part of the experiments. We randomly generated 100 instances for each situation and we reported the results in Table 6. The mean and standard deviation of the relative deviation percentage (RDP) were given. The RDP of the

<table>
<thead>
<tr>
<th>$n$</th>
<th>$P$</th>
<th>$\tau$</th>
<th>$R$</th>
<th>SA$_1$ Mean</th>
<th>SA$_1$ SD</th>
<th>SA$_2$ Mean</th>
<th>SA$_2$ SD</th>
<th>CPU time SA$_1$ Mean</th>
<th>SA$_2$ Mean</th>
<th>SA$_1$ SD</th>
<th>SA$_2$ SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.25</td>
<td>0.50</td>
<td>0.25</td>
<td>1.89</td>
<td>0.25</td>
<td>3.07</td>
<td>0.20</td>
<td>0.23</td>
<td>0.23</td>
<td>1.20</td>
<td>0.21</td>
</tr>
<tr>
<td>0.75</td>
<td>0.50</td>
<td>0.25</td>
<td>2.92</td>
<td>0.24</td>
<td>0.25</td>
<td>2.92</td>
<td>0.24</td>
<td>0.25</td>
<td>0.25</td>
<td>1.30</td>
<td>0.22</td>
</tr>
<tr>
<td>0.75</td>
<td>0.50</td>
<td>0.25</td>
<td>2.92</td>
<td>0.24</td>
<td>0.25</td>
<td>2.92</td>
<td>0.24</td>
<td>0.25</td>
<td>0.25</td>
<td>1.30</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 6: The performance of the SA$_1$ and SA$_2$ algorithms with $\theta = 0.5$.  


solution produced by a simulated annealing algorithm is calculated as

\[
\frac{(V_i - \min \{V_1, V_2\})}{\min \{V_1, V_2\}} \times 100\% 
\] (13)

for \(i = 1, 2\), where \(V_i\) is the value of the weighted combination of the total completion time and the total tardiness of the jobs from the first agent generated by the \(i\)th simulated annealing algorithm. The mean and standard deviation of the execution time were also reported. In addition, we recorded the number of times \((n_2)\) it yields the minimum value. It was seen that both algorithms are very fast since the mean execution time is less than 1 second for all the tested cases. However, the first SA algorithm performs better than the second one in terms of the RDP values and the number of times it yields the minimum value. Thus, it is recommended as the number of jobs increases.

### 6. Conclusion

In this paper, we considered a two-agent single-machine scheduling problem where the objective is to minimize the weighted combination of the total completion time and the total tardiness of the jobs of the first agent given that no tardy job is allowed for the second agent. We proposed a branch-and-bound algorithm to solve the problem optimally and two simulated annealing algorithms to find near-optimal solutions. The computational experiments showed that the branch-and-bound algorithm could solve problems of up to 24 jobs in a reasonable amount of time. It also showed that the performance of the first SA algorithm is very good, yielding an average error percentage of less than 1% for all the tested cases.

### Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

### Acknowledgments

The authors are grateful to the referees for their comments. This work is supported by the National Science Council of Taiwan, under Grant no. NSC 1000-2221-E-035-029-MY3.

### References


