J. Soc. Korea Ind. Syst. Eng Vol. 38, No. 3 : 169-180, September 2015
http://dx.doi.org/10.11627/jkise.2015.38.3.169
ISSN : 2005-0461(print)
ISSN : 2287-7975(online)

Two-Agent Single-Machine Scheduling with Linear Job-Dependent Position-Based Learning Effects

Jin Young Choi†

Department of Industrial Engineering, Ajou University

작업 종속 및 위치기반 선형학습효과를 갖는 2-에이전트 단일기계 스케줄링

최 진 영†

아주대학교 산업공학과

Recently, scheduling problems with position-dependent processing times have received considerable attention in the literature, where the processing times of jobs are dependent on the processing sequences. However, they did not consider cases in which each processed job has different learning or aging ratios. This means that the actual processing time for a job can be determined not only by the processing sequence, but also by the learning/aging ratio, which can reflect the degree of processing difficulties in subsequent jobs. Motivated by these remarks, in this paper, we consider a two-agent single-machine scheduling problem with linear job-dependent position-based learning effects, where two agents compete to use a common single machine and each job has a different learning ratio. Specifically, we take into account two different objective functions for two agents: one agent minimizes the total weighted completion time, and the other restricts the makespan to less than an upper bound. After formally defining the problem by developing a mixed integer non-linear programming formulation, we devise a branch-and-bound (B&B) algorithm to give optimal solutions by developing four dominance properties based on a pairwise interchange comparison and four properties regarding the feasibility of a considered sequence. We suggest a lower bound to speed up the search procedure in the B&B algorithm by fathoming any non-prominent nodes. As this problem is at least NP-hard, we suggest efficient genetic algorithms using different methods to generate the initial population and two crossover operations. Computational results show that the proposed algorithms are efficient to obtain near-optimal solutions.

Keywords : Two-Agent Single-Machine Scheduling, Job-Dependent Position-Based Processing Time, Total Weighted Completion Time, Makespan, Branch-and-Bound Algorithm, Genetic Algorithm

1. Introduction

In this paper, we consider a scheduling problem in which two agents compete to use a common single machine, and each agent has different performance objectives when processing a set of jobs. The processing order of one job belonging to one agent can affect the objective function of the other agent. Therefore, we need to find a schedule that optimizes two objective functions at the same time by considering the impact of scheduled jobs on the objective functions of two agents.

This scenario is called a two-agent scheduling problem, and requires special attention. If we handle this problem as

Received 6 August 2015; Finally Revised 17 September 2015; Accepted 17 September 2015
† Corresponding Author : choijy@ajou.ac.kr

a simple bi-criteria, single-agent optimization problem, we could use one of two general approaches : (i) represent the objective function as the weighted sum of two objective functions, or (ii) consider two objective functions one-by-one, using the first result as a new constraint for the second. However, neither of these methods can guarantee the optimality of the obtained solution, as the two objective functions might have different measurement units, and all jobs contribute to all criteria [2]. Therefore, two-agent scheduling problems are different from those commonly referred to as bi-criteria scheduling problems. The complexity of two-agent scheduling problems is higher than that of bi-criteria scheduling problems, and we need to devise more efficient and systematic approaches.

Two-agent scheduling problems occur in various industrial environments where two agents interact to perform their respective tasks using a common processing resource. For example, let us consider a project scheduling problem in a firm. As explained in [15], over time, multiple projects within a factory might compete to use shared common renewable resources, such as people and machinery. Each project manager is responsible for achieving good performance on his or her project, and so the use of these resources must be negotiated with other project managers. Another example can be found when tasks in a mechanical workshop require rescheduling [26]. If the main objective is to minimize the total flow time, some jobs that have not been done on one day become urgent the next. Two groups of jobs can then be assigned to two agents with different objective functions.

In recent decades, there have been a number of studies on this issue. First, the concept of two-agent scheduling problems was introduced by [4], which analyzed the computational complexity of combining two criteria into a single objective function. Then, Agnetis et al. [1] addressed the two-agent scheduling problem by considering the makespan, lateness, sum of completion times, and sum of weighted completion times, and analyzed the problem complexity while suggesting some solution algorithms. Ng et al. [25] considered the case whereby one agent minimizes the sum of completion times, while restricting the number of tardy jobs within a bound for the other agent, and showed that this problem is NP-hard under the condition of high multiplicity. Cheng et al. [9] extended the work of Agnetis et al. [1] to multi-agent scheduling problems. Readers who are interested in this topic are referred to [11, 18, 32], and the references therein.

The above studies did not consider cases in which the job processing time depends on the position (or start time) of a job within the overall sequence. We call this the position-dependent processing time. In the literature, this processing time is categorized as having a learning effect or an aging effect. With a learning effect, the actual processing time of a job decreases according to its position in the sequence. For example, repeated processing of similar tasks improves workers' skills, increasing the speed of machine operation. With an aging effect, the actual processing time of a job increases with its position in the sequence. As a practical example, we can consider the continuous casting and rolling processes in a steel plant, where the hot and cold charge process can be regarded as two agents - the processing time of the charges will increase with time due to the drop in temperature [20].

Recently, scheduling problems with position-dependent processing times have received considerable attention in the literature. Specifically, Biskup [5] introduced the concept of the learning effect into scheduling problems for a single agent. Mosheiov [24] mentioned the aging effect, and showed that a V-shaped schedule is optimal for the single-machine flowtime minimization problem [23]. Biskup [6] provided an extensive survey on scheduling problems with learning effects, and Kuo and Yang [16] considered a single-machine scheduling problem with a cyclic aging effect. Chang et al. [7] studied the learning/aging effect scheduling problem on a single machine with a common due date. For more recent results, see [30, 27, 33].

To the best of our knowledge, few studies on scheduling problems have simultaneously considered two-agent and position (or start time)-dependent processing times. These studies can be categorized by the functional representation method they use to compute the actual processing times : (i) linear function-based approaches (e.g., [20, 21, 17, 31]), and (ii) exponential function-based approaches (e.g., [10, 19, 29]). The above studies considered different combinations of objective functions, such as the (weighted) sum of completion times, (weighted) sum of tardiness, or lateness for one agent, under a constraint on the number of tardy jobs or the upper bound of the makespan for the other agent.

However, they did not consider cases in which each processed job has different learning or aging ratios. This means that the actual processing time for a job can be determined not only by the processing sequence, but also by the learning/aging ratio, which can reflect the degree of processing difficulties in subsequent jobs. This modeling concept for the processing time was first suggested by Cheng and Wang [8]. Later, Bachman and Janiak [3] analyzed a simpler learning effect formulation for a single-machine case, and Wang

and Xia [28] applied the concept to multiple-machine flowshop scheduling with an increasing series of dominating machines. These are plausible scenarios in real-life scheduling environments, and hence they deserve to be applied to the case of two-agent single-machine scheduling.

Thus, in this paper, we consider a two-agent single-machine scheduling problem with linear job-dependent position-based learning effects, where each job has a different learning ratio. Specifically, we take into account two objective functions such that one agent wants to minimize the total weighted completion time and the other agent wishes to restrict the makespan to within some upper bound. We formally define the problem by developing a mathematical formulation, and devise a branch-and-bound (B&B) algorithm to give optimal solutions. As this problem is at least binary NP-hard, we suggest efficient genetic algorithms (GAs) using different methods to generate the initial population and two crossover operations to find near-optimal solutions, and verify their performance by means of a numerical experiment.

The rest of this paper is organized as follows. In Section 2, we formally define the two-agent single-machine scheduling problem with linear job-dependent position-based learning effects by developing an appropriate mixed integer programming (MIP) model and design a B&B algorithm to find optimal solutions. In Section 3, we develop efficient GAs that give near-optimal solutions and verify the performance of these algorithms using a numerical experiment in Section 4. Finally we give our conclusions and suggestions for future work in Section 5.

2. Problem Definition and a Branch-and-Bound Algorithm

2.1 Problem Definition

The scheduling problem considered can be described as follows : Consider two agents A and B, each with a set of non-preemptive n_X jobs $J^X=\{J_1^X, J_2^X, \cdots, J_{n_X}^X\}$, where J_i^X represents the ith job of agent X, for $X\in\{A, B\}$. Any of these jobs can be processed at the beginning of the operation, and each of them is processed by a single common machine. Therefore, the two agents compete to use the machine while optimizing their own objective functions, which are dependent on the completion times of all jobs. Specifically, we assume that agent A is aiming to minimize the sum of the weighted completion times of all jobs in J^A, represented by $\sum_{i=1}^{n_A} w_i^A C_i^A$, and that agent B must ensure that the makespan $C_{\max}^B$ is less than a given upper bound U, where w_i^A and C_i^A are the weight and the completion time of job J_i^A, respectively, for $i=1, 2, \cdots, n_A$.

Based on this problem description, we consider a special model for processing times that are position-dependent linear functions with job-dependent learning effects. Thus, for agent $X\in\{A, B\}$, the actual processing time of job J_i^X processed in rth position in a sequence, represented by $p_i^X(r)$, has a learning effect of $p_i^X(r)=p_i^X-rb_i^X$, where p_i^X is the normal processing time of job J_i^X and $b_i^X>0$ is the constant learning ratio of job J_i^X, for $i\in I^X=\{1, 2, \cdots, n_X\}$ and $r\in I=\{1, 2, \cdots, n\}\,(n=n_A+n_B)$. Different schedules may therefore give different values $\sum_{i=1}^{n_A} w_i^A C_i^A$ and $C_{\max}^B$ for the two agents. Specifically, for a schedule S, we represent these as $\sum_{i=1}^{n_A} w_i^A C_i^A(S)$ and $C_{\max}^B(S)$, where $C_{\max}^B(S)$ can be computed as $C_{\max}^B(S)=\max_{k\in I^B} C_k^B(S)$ using $C_k^B(S)$ to represent the completion time of J_k^B, $k\in I^B$. Furthermore, because all processing times are positive, we assume that

$$b_i^X<\frac{p_i^X}{n},\ \forall J_i^X,\ i\in I^X,\ X\in\{A, B\}.$$

Using the three-field notation $\Psi_1|\Psi_2|\Psi_3$, suggested by [12], we can denote the scheduling problem as

$$1|p_i^A(r)=p_i^A-rb_i^A, p_j^B(v)=p_j^B-vb_j^B \\ |\sum_{i=1}^{n_A} w_i^A C_i^A: C_{\max}^B\le U, \tag{1}$$

where the first component Ψ_1 is the number of machines, Ψ_2 describes the job characteristics, and Ψ_3 contains the objective functions. Therefore, Eq. 1 represents the two-agent single-machine scheduling problem with linear job-dependent position-based learning effects.

Furthermore, by defining

$$J_j=\begin{cases} J_j^A, & \text{if } 1\le j\le n_A \\ J_{j-n_A}^B, & \text{if } n_A+1\le j\le n_A+n_B \end{cases},$$

we can develop the following MIP model for the problem under consideration :

$$Minimize\ Z_1 = \sum_{j=1}^{n_A} w_j^A C_j \tag{2}$$

$s.t.$

$$p_j = \begin{cases} \sum_{r=1}^{n} \left(p_j^A - r b_j^A\right) x_{jr}, & \text{if } 1 \le j \le n_A \\ \sum_{v=1}^{n} \left(p_j^B - r b_j^B\right) x_{jv}, & \text{if } n_A+1 \le j \le n \end{cases} \tag{3}$$

$$C_{[r]} = \sum_{l=1}^{r} \sum_{j=1}^{n} p_j x_{jl},\ r = 1, 2, \cdots, n \tag{4}$$

$$C_j = \sum_{r=1}^{n} C_{[r]} x_{jr},\ j = 1, 2, \cdots, n \tag{5}$$

$$C_j \le U,\ j = n_A+1, n_A+2, \cdots, n \tag{6}$$

$$\sum_{r=1}^{n} x_{jr} = 1,\ j = 1, 2, \cdots, n \tag{7}$$

$$\sum_{j=1}^{n} x_{jr} = 1,\ r = 1, 2, \cdots, n \tag{8}$$

$$C_j \ge 0,\ j = 1, 2, \cdots, n \tag{9}$$

$$C_{[r]} \ge 0,\ r = 1, 2, \cdots, n \tag{10}$$

$$x_{jr} \in \{0, 1\},\ j = 1, 2, \cdots, n,\ r = 1, 2, \cdots, n, \tag{11}$$

where x_{jr} is defined as $x_{jr} = 1$ if job J_j is assigned to the rth position and $x_{jr} = 0$, otherwise. C_j is the completion time of job J_j, p_j is the processing time of job J_j, and $C_{[r]}$ is the completion time of the job in position r. Eq. 2 therefore represents the total weighted completion time of agent A, which is the objective function of this agent, and Eq. 3 denotes the position-dependent processing time of job J_j under consideration of learning effects. Eq. 4 computes the completion time of the rth job, and the completion time of job J_j can be obtained by using this in Eq. 5. Eq. 6 is the upper-bound makespan condition for agent B. Eqs. 7 and 8 are job assignment conditions: each job must be assigned to only one position, and each position can have only one job. Eqs. 9 and 10 are non-negativity conditions for variables C_j and $C_{[r]}$, and Eq. 11 is the binary condition for variable x_{jr}.

Agnetis et al. [1] showed that a two-agent single-machine scheduling problem $1 \| \sum_{i=1}^{n_A} w_i^A C_i^A : f_{\max}^B$ is binary NP-hard, even without job learning or aging effects. Therefore, our problem is at least binary NP-hard, necessitating the development of an efficient solution procedure to find a (near-) optimal solution.

2.2 Properties for Dominance and Feasibility

First, we develop four dominance properties based on a pairwise interchange comparison. Suppose that there are two sequences $s_1 = (\pi_1, J_i^X, J_j^X, \pi_2)$ and $s_2 = (\pi_1, J_j^X, J_i^X, \pi_2)$, where π_1 and π_2 represent the scheduled and unscheduled parts, respectively. In S_1, two jobs J_i^X and J_j^X are in the rth and $(r+1)$th positions, respectively, whereas S_2 is obtained by interchanging J_i^X and J_j^X in S_1. Then, by defining t as the completion time of sequence π_1, we have the following properties.

(Property 1)

If two jobs $J_i^X, J_j^X \in J^A$ satisfy (i) $b_i^A < b_j^A$ and (ii) $w_j^A / w_i^A \le \left(p_j^A(r) - b_i^A\right) / \left(p_i^A(r) - b_j^A\right)$, then S_1 dominates S_2.

Proof : From the definition, we can compute the completion times of J_i^A and J_j^A for schedules S_1 and S_2 as follows.

$$\begin{aligned} C_i^A(S_1) &= t + \left(p_i^A - r b_i^A\right) \\ C_j^A(S_1) &= t + \left(p_i^A - r b_i^A\right) + \left(p_j^A - (r+1) b_j^A\right) \\ C_j^A(S_2) &= t + \left(p_j^A - r b_j^A\right) \\ C_i^A(S_2) &= t + \left(p_j^A - r b_j^A\right) + \left(p_i^A - (r+1) b_i^A\right). \end{aligned}$$

From $b_i^A < b_j^A$, we have $C_j^A(S_1) < C_i^A(S_2)$. Since $w_j^A / w_i^A \le \left(p_j^A(r) - b_i^A\right) / \left(p_i^A(r) - b_j^A\right)$, we have $w_i^A C_i^A(S_1) + w_j^A C_j^A(S_1) \le w_j^A C_j^A(S_2) + w_i^A C_i^A(S_2)$. Therefore, S_1 can have a smaller total weighted completion time for agent A and a faster makespan for agent B than S_2. This means that S_1 dominates S_2. ■

Three further properties can be proved in a similar manner to Property 1.

(Property 2)

Given two jobs $J_i^X, J_j^X \in J^B$, if (i) $b_i^B < b_j^B$ and (ii) $t + \left(p_i^B - r b_i^B\right) + \left(p_j^B - (r+1) b_j^B\right) \le U$, then S_1 dominates S_2.

(Property 3)

Given two jobs $J_i^X \in J^A$, $J_j^X \in J^B$, if (i) $b_i^A < b_j^B$ (ii) $p_j^B(r) - b_i^A \ge 0$ and (iii) $t + \left(p_i^A - r b_i^A\right) + \left(p_j^B - (r+1) b_j^B\right) \le U$, then S_1 dominates S_2.

(Property 4)

Given two jobs $J_i^X \in J^B, J_j^X \in J^A$, if (i) $b_i^B < b_j^A$ (ii) $p_i^B(r) - b_j^A \ge 0$ and (iii) $t + \left(p_i^B - r b_i^B\right) \le U$, then S_1 dominates S_2.

Additionally, we can state the following four properties regarding the feasibility of a sequence. Suppose that we have a sequence of jobs (π, π^c), where the scheduled part π consists of m jobs and the unscheduled part π^c consists of $(n-m)$ jobs. Furthermore, let $\tilde{p}(m+r)$ be the minimum actual processing time of jobs that can be scheduled in the rth position in π^c, which is position $(m+r)$ in the whole sequence.

(Property 5)
Given a job $J_i^X \in J^B \cap \pi$ scheduled in the mth position with $C_{[m]} > U$, then (π, π^c) is infeasible.

Proof : Since job $J_i^X \in J^B \cap \pi$ in the mth position has $C_{[m]} > U$, it violates the condition for agent B. ■

(Property 6)
Given a job $J_i^X \in J^A \cap \pi$ scheduled in the mth position with $C_{[m]} > U$ and a job $J_j^X \in J^B \cap \pi^c$, then (π, π^c) cannot generate any feasible sequence.

Proof : Since job $J_i^X \in J^A \cap \pi$ and $C_{[m]} > U$, the completion time of any job $J_j^X \in J^B \cap \pi^c$ must be greater than U. Therefore, any sequence generated from (π, π^c) is not feasible. ■

(Property 7)
Given a job $J_i^X \in J^B \cap \pi^c$, $C_{[m]} \le U$, and $C_{[m]} + \tilde{p}(m+1) > U$, then (π, π^c) cannot generate any feasible sequence.

Proof : By the definition of $\tilde{p}(m+1)$, we have $p_i^B(v) > \tilde{p}(m+1)$ for any job $J_i^X \in J^B \cap \pi^c$ and any position v in π^c. Therefore, we have $C_i^B > U$ for any job $J_i^X \in J^B \cap \pi^c$, making any schedule generated from (π, π^c) infeasible. ■

(Property 8)
If all jobs in π^c belong to agent B, then sequence (π, π^c) should be $(\pi, \tilde{\pi})$, where $\tilde{\pi}$ is the sequence obtained by scheduling jobs in π^c in non-decreasing order of b_j^B.

Proof : Since all jobs in π^c belong to agent B, we have a scheduling problem with $(n-m)$ jobs for agent B, represented by

$$1|p_j^B(v) = p_j^B - vb_j^B|C_{\max}^B. \tag{12}$$

According to Bachman and Janiak [3], this problem can be optimized by scheduling jobs in non-decreasing order of b_j^B. ■

2.3 Branch-and-Bound Algorithm

Using the properties stated above, we now develop an efficient B&B algorithm. The algorithm attempts to assign jobs in a forward manner, i.e., iteratively from the first position. The basic idea is to branch a node into several nodes, each corresponding to the scheduling of one job among all possible jobs, and to bound each of them by computing the potential minimum value (i.e., lower bound) of the total weighted completion time of agent A for the schedule under consideration [14]. For each node, we perform the following process.

At each iteration of the algorithm, suppose that we have a sequence of jobs $S = (\pi, \pi^c)$, where the unscheduled part π^c comprises n_1 jobs for agent A and n_2 jobs for agent B. Then, we have

$$C_{[m+1]} = C_{[m]} + p[m+1] \ge C_{[m]} + \tilde{p}(m+1),$$
$$C_{[m+l]} \ge C_{[m]} + \sum_{r=1}^{l} \tilde{p}(m+r), \tag{13}$$

where $p[m+1]$ is the actual processing time of the $(m+1)$th job. Therefore, we can use Eq. 13 as the estimator of $C_{[m+l]}$, represented by

$$\hat{C}_{[m+l]} = C_{[m]} + \sum_{r=1}^{l} \tilde{p}(m+r).$$

We can also sort the weights of the unscheduled n_1 jobs into non-increasing order, as $w_{(1)}^A \ge w_{(2)}^A \ge \cdots \ge w_{(n_1)}^A$. Then, the lower bound of $\sum_{i=1}^{n_A} w_i^A C_i^A(S)$ can be computed as follows.

(Lemma 1)
For schedule S with (π, π^c), the lower bound of $\sum_{i=1}^{n_A} w_i^A C_i^A(S)$ at the current iteration is

$$LB(S) = \sum_{r=1}^{n_A - n_1} w_{[r]}^A C_{[r]}^A + \sum_{r=1}^{n_1} w_{(r)}^A C_{(r)}^A, \tag{14}$$

where $w_{[r]}^A$ and $C_{[r]}^A$ represent the weight and completion time of the job scheduled in rth position among all $(n_A - n_1)$ jobs for agent A in π, respectively. Furthermore, $C_{(r)}^A$ denotes the minimum completion time of a job scheduled in rth position among all n_1 jobs for agent A in π_c.

Proof : The first term in Eq. 14 computes the total weighted completion time of $(n_A - n_1)$ scheduled jobs for agent A in π. According to [13], $\sum_{i=1}^{n} a_i b_i$ is minimized if the two sequences of numbers a_i and b_i are monotonic in the opposite sense. Based on this, the minimum value of the total weighted completion time that can be achieved using the unscheduled n_1 jobs for agent A in π^c is $\sum_{r=1}^{n_1} w_{(r)}^A C_{(r)}^A$, which gives the second term in Eq. 14, resulting in a lower bound LB for S. ■

Specifically, we can compute C_r^A by assigning the estimated completion times $\hat{C}_{[m+l]}$ $(l = 1, 2, \cdots, n_1 + n_2)$ to the remaining $(n_1 + n_2)$ jobs for agents A and B as follows :

- Step 1 : Set $tot = 0$, $ca = n_1$, and $cb = n_2$.
- Step 2 : If $tot \geq n_1 + n_2$, or $ca = 0$, or $cb = 0$, go to Step 8.
- Step 3 : If $\hat{C}_{[n-tot]} \leq U$ and $cb > 0$, set $C_{(ab)}^B = \hat{C}_{[n-tot]}$, $cb = cb - 1$, update U as $U - \tilde{p}(n - tot)$, and go to Step 7. Otherwise, go to Step 4.
- Step 4 : If $ca > 0$, set $C_{(ca)}^A = \hat{C}_{[n-tot]}$, $ca = ca - 1$ and go to Step 5. Otherwise, go to Step 6.
- Step 5 : If $cb < n_2$, update U as $U - \tilde{p}(n - tot)$. Go to Step 7.
- Step 6 : If $cb > 0$, set $C_{(cb)}^B = \hat{C}_{[n-tot]}$ and $cb = cb - 1$.
- Step 7 : Set $tot = tot + 1$ and go to Step 2.
- Step 8 : We have $C_{(r)}^A$ for $r = 1, 2, \cdots, n_1$.

Since the lower bound gives the minimum value of the total weighted completion time for agent A that can be achieved after finishing the allocation procedure, it can be used to speed up the search procedure in the B&B algorithm by fathoming any non-prominent nodes. We apply a depth-first search to the B&B algorithm, the overall procedure of which can be described as follows.

- Initialization : Set $Z^* = \infty$ and the current level $L = 0$. Apply the bounding step and fathoming step described below to the whole problem. If it is not fathomed, consider this as the one remaining subproblem for the iteration below.
- Steps for each iteration :
  1. *branching* : If there are no remaining subproblems at the current level L, set $L \leftarrow L - 1$ and go to the optimality test step. Otherwise, choose the subproblem with the best lower bound LB among those remaining at the current level L. If more than one subproblem have the lowest value of LB, select one at random. Branch from the node to create new subproblems as follows:
     (a) If all remaining jobs belong to agent B, then *apply Property* 8 to generate one subproblem corresponding to the scheduling of all remaining jobs for agent B.
     (b) Otherwise, create subproblems corresponding to the case of scheduling one remaining, non-dominated (by *Properties* 1 to 4), and feasible (by *Properties* 5 to 7) job in the $(m+1)$th position, where m is the last position of the scheduled part for the selected node.
     (c) For each new subproblem, update the set of scheduled jobs and unscheduled jobs for agents A and B.
  2. *bounding* : For each new subproblem, compute $C_{[m]}$ and LB using Eq. 14.
  3. *Fathoming* : For each new subproblem, apply the following two fathoming tests. A subproblem with a sequence of jobs (π, π^c) is fathomed if
     (a) Test 1 : $LB > Z^*$, or
     (b) Test 2 : There are no remaining jobs to be scheduled (If $LB < Z^*$, this becomes the new value of Z^*, and Test 1 is reapplied to all unfathomed subproblems with this new Z^*).
     (c) Set $L \leftarrow L + 1$.
- Optimality test : Stop and accept the current solution as optimal if there are no remaining subproblems or if $L = 0$. Otherwise, perform another iteration.

3. An Efficient GA

It is well known that B&B algorithms require considerable computational time to find optimal solutions to large-sized or computationally difficult problems. Therefore, we develop a GA Mitchell [22], a well-known meta-heuristic approach that can find near-optimal solutions efficiently. The components of the suggested GA are as follows.

- Design of a chromosome : First, we design a chromosome P as an n-dimensional array representing a sequence of n jobs for two agents. Each gene represents the job number, and its position in the chromosome corresponds to the order of the jobs in a schedule. Then, we can make the following statement on the validity of the suggested structure for a chromosome.

(Property 9) Feasibility condition for a chromosome
A chromosome is valid and feasible as a sequence for the scheduling problem under consideration if all jobs represented by genes satisfy the following two conditions : (i) job numbers are not duplicated, and (ii) the makespan for agent B satisfies the upper-bound condition.

- Initialization of a population : We consider three methods of generating a chromosome to initialize a population :
  1. IP_1 : Arrange the jobs for agent B in non-decreasing order of b_j^B, then schedule the jobs for agent A in order of shortest normal processing time.
  2. IP_2 : Schedule the jobs for agent B in non-decreasing order of b_j^B, then arrange the jobs for agent A in order of weighted shortest normal processing time.
  3. IP_3 : Create a chromosome at random. If it does not satisfy Property 9, regenerate it.

We make Q chromosomes for the initial population, where Q is an even number. During the GA procedure, we allow a modest number of duplications, and set $Q=30\times n$ to increase the diversity of the chromosomes.

- Fitness cost function : We use the total weighted completion time of jobs for agent A as the fitness cost.
- Parent selection : Chromosomes in the population are sorted in ascending order of fitness cost, and are then paired, starting from those with the smallest fitness cost. Each pair generates two offspring by applying the following reproduction process.
- Reproduction : Reproduction generates offspring that have the parents' characteristics. First, we use a crossover operation, exchanging some parts of two chromosomes. Specifically, we consider two kinds of crossover operation : (i) CO_1 : one-point crossover, and (ii) CO_2 : two-point crossover. However, in general, any crossover operation applied to a scheduling problem encounters a feasibility issue in the resulting chromosome, where there might be some duplicated genes. Therefore, we design a special method that guarantees the feasibility of the resulting chromosome. Suppose that we have two parents P_1 and P_2. One-point (two-point) crossover randomly selects one point (two points) on P_1, and reorders the genes between the selected first gene and the last (second) gene according to the sequence in P_2, producing a new offspring C_1. If the offspring is not feasible, we reapply this procedure. Similarly, we can generate another offspring C_2. For each pair of chromosomes, we apply this operation probabilistically according to some pre-specified crossover rate $r_c(0<r_c<1)$. Otherwise, we generate two offspring as $C_1=P_1$ and $C_2=P_2$.

Moreover, we apply a mutation operation to each chromosome by swapping two randomly selected genes. This operation is useful in escaping from local optima, and is applied probabilistically according to some pre-specified rate $r_m(0<r_m<1)$. If the resulting chromosome is not feasible under Property 9, we repeat the procedure.

- Reconstruction of a population : By applying the reproduction procedure to each pair of chromosomes, we can generate Q offspring. After evaluating the fitness cost of all offspring, we have a total of $2\times Q$ chromosomes. Then, we can select the best Q chromosomes to reconstruct a new population of the same size.
- Termination condition : We terminate the GA if one of the following conditions is satisfied : (i) the maximum number of iterations has been reached, or (ii) there is no improvement in the solution for a fixed number of consecutive iterations.
- Overall procedure of the suggested GA : The overall procedure of the suggested GA consists of three parts:
  1. *Initialization* : Initialize the GA parameters such as the crossover probability r_c, mutation probability r_m, and termination conditions. Start with an initial population of size $Q=30\times n$, and evaluate the fitness cost for each member of the population. Sort chromosomes in ascending order of fitness, and identify that which has the minimum cost in the current population.
  2. Iteration procedure : Pair up the chromosomes in order of fitness cost values, from smallest to largest, and apply the reproduction procedure. Evaluate the fitness costs of the generated offspring. Select the best Q chromosomes among the newly generated members and the

current population, and form a new population for the next generation. Sort the population in ascending order of fitness cost values, and identify the solution with the minimum fitness cost.

3. Stooping rule : The GA stops with the best trial solution found so far when a fixed number of consecutive iterations do not result in any improvement. Otherwise, the procedure is repeated for the next generation.

4. Numerical Experiments

4.1 Experimental Design

To verify the performance of the GA described in the previous section, we designed a numerical experiment. First, we considered different scheduling problems by changing the number of jobs for each agent. Specifically, we considered $n=10, 12, 14, 16$, assuming that each agent has the same number of jobs. The value of U for agent B was generated using the convex combination of the minimum value of the makespan and the minimum value of the maximum makespan, represented by V_1 and V_2, respectively. The value of V_1 can be obtained by solving the single-agent scheduling problem in Eq. 12. V_2 can be computed by the following lemma.

(Lemma 2)
Let $C^B_{\max,J_k^B}$ denote the makespan of a schedule S with J_k^B in the last position. Then, we have

$$\max C^B_{\max,J_k^B} = \left(\sum_{i=1}^{n_A} p_i^A + \sum_{\substack{j=1 \\ j\neq k}}^{n_B} p_j^B\right) - \min \sum_{\substack{i=1 \\ i\neq n_A+k}}^{n} \sum_{r=1}^{n-1} rb_i x_{ir} + \left(p_k^B - nb_k^B\right),$$

and $V_2 = \min_{k\in I^B} \max C^B_{\max,J_k^B}$,

where $b_i = b_i^A$, if $1 \le i \le n_A$ and $b_i = b^B_{i-n_A}$, if $n_A+1 \le i \le n_A+n_B$,

Proof : For any schedule S' obtained from schedule S by removing J_k^B, we have

$$\begin{aligned}
&\max C^B_{\max,J_k^B} = \max_{S'} C_{\max}(S') + p_k^B(n) \\
&= \max\left(\begin{array}{c}\sum_{i=1}^{n_A} p_i^A + \sum_{\substack{j=1 \\ j\neq k}}^{n_B} p_j^B \\ - \sum_{\substack{i=1 \\ i\neq n_A+k}}^{n} \sum_{r=1}^{n-1} rb_i x_{ir}\end{array}\right) + \left(p_k^B - nb_k^B\right) \\
&= \left(\sum_{i=1}^{n_A} p_i^A + \sum_{\substack{j=1 \\ j\neq k}}^{n_B} p_j^B\right) - \min \sum_{\substack{i=1 \\ i\neq n_A+k}}^{n} \sum_{r=1}^{n-1} rb_i x_{ir} + \left(p_k^B - nb_k^B\right)
\end{aligned}$$

where $C_{\max}(S')$ is the makespan of the remaining $(n-1)$ jobs using schedule S'. Furthermore, we can compute $\min \sum_{\substack{i=1 \\ i\neq n_A+k}}^{n} \sum_{r=1}^{n-1} rb_i x_{ir}$ by arranging the remaining jobs in non-increasing order of b_i^A and b_j^B. ■

Now, we can obtain different values of U by defining

$$U = \alpha V_1 + (1-\alpha) V_2, \tag{15}$$

where we consider the three different values of $\alpha =$ 0.25, 0.5 and 0.75. Therefore, we have 12 system configurations defined by different pairs of (n, α). For each configuration, we produced 50 problem instances by randomly generating normal processing times in the range 1-100, learning effects in the range 1-100, and job weights in the range 1-100. Accordingly, we considered 4×3×50 = 600 problem instances.

Moreover, we considered three methods of generating the initial population and two different reproduction operations. Hence, we have 3×2 = 6 different GAs, represented by $GA(IP_k, CO_t)(k=1, 2, 3,\ t=1, 2)$. We determined the parameter values of these GAs by pretesting some randomly generated problem instances. As a result, we set $r_c =$ 0.8, $r_m =$ 0.1, and applied the stopping condition if the GA went 30 iterations without any improvement. Each instance of the scheduling problem was solved using the B&B algorithm and the six GAs.

The performance of the GAs was evaluated in terms of the percentage error, which is defined as

$$\%\,error = \frac{TWCT\ by\ GA - Optimal\ TWCT}{Optimal\ TWCT} \times 100$$

where $TWCT$ is the total weighted completion time. Furthermore, we measured the number of generated nodes and

CPU time using the B&B algorithm. The CPU time of the B&B algorithm was compared to that required by the GAs. To compare the six GAs, we defined the relative deviation percentage (RDP) as

$$RDP = \left(\frac{TWCT\,by\,GA}{Minimum\ TWCT\,by\,any\ GA} - 1 \right) \times 100 \quad (16)$$

4.2 Experimental Results and Assessment

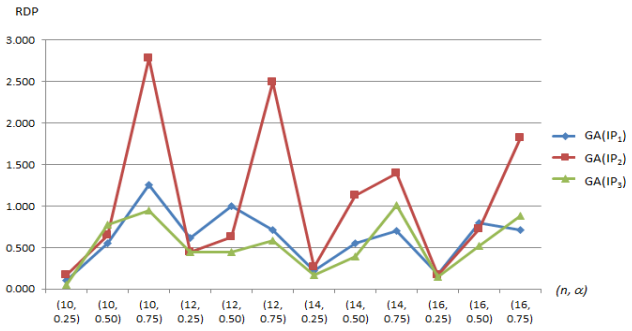
Our numerical experiments considered the solutions to 600 problem instances. For each system configuration and solution method, we calculated the mean, standard deviation (stdev), maximum number of generated nodes (for the B&B), CPU time (in seconds), and % error. <Table 1> shows the experimental results. B&B could solve instances with up to 14 jobs with a reasonable CPU time. Even when $n = 16$, it solved instances generated using $\alpha = 0.25$ efficiently. For a fixed value of n, B&B took more time to obtain the optimal solution as the value of was increased. This is because, from Eq. 15, small values of α increase the value of U, since $V_1 \leq V_2$. Similarly, larger values of α make U smaller, giving a tight upper bound U for agent B, and making it difficult to find an optimal solution using B&B.

Overall, the GAs exhibited good performance in the sense that they gave small % errors, with a mean of less than 1% in almost every configuration. Even for the largest configuration $(n, \alpha) = (16, 0.75)$, the maximum % error (using IP_2)

<Table 1> Results of Numerical Experiments (CPU time in s)

n	α	Value of	BB		$GA(IP_1, CO_1)$		$GA(IP_1, CO_2)$		$GA(IP_2, CO_1)$		$GA(IP_2, CO_2)$		$GA(IP_3, CO_1)$		$GA(IP_3, CO_2)$	
			Nodes	CPU time	% error	CPU time	% error	CPU time	% error	CPU time	% error	CPU time	% error	CPU time	% error	CPU time
10	0.25	min	3584.720	1.018	0.096	0.783	0.113	0.770	0.137	0.767	0.192	0.783	0.038	0.581	0.059	0.582
		stdev	11098.810	2.958	0.291	0.116	0.302	0.099	0.398	0.107	0.591	0.098	0.147	0.099	0.264	0.099
		max	77781.000	20.545	1.329	1.201	1.654	0.967	1.986	1.045	3.290	1.046	0.861	0.843	1.627	0.905
	0.50	min	6748.140	2.134	0.220	0.763	0.048	0.764	0.231	0.773	0.111	0.755	0.280	0.609	0.040	0.599
		stdev	9815.683	3.227	0.671	0.100	0.211	0.101	0.933	0.102	0.575	0.081	0.980	0.092	0.165	0.089
		max	58041.000	19.563	3.382	1.107	1.072	1.139	5.242	1.045	4.030	0.967	4.896	0.858	1.072	0.827
	0.75	min	6345.680	2.376	0.282	0.705	0.474	0.693	1.684	0.681	1.634	0.688	0.401	0.741	0.104	0.744
		stdev	5608.517	2.122	1.115	0.102	1.654	0.106	3.957	0.132	4.154	0.116	1.404	0.087	0.735	0.105
		max	26935.000	10.140	5.640	0.983	7.787	1.045	17.098	1.061	17.158	0.968	5.936	0.905	5.197	1.014
12	0.25	min	32307.720	10.124	0.251	1.247	0.389	1.200	0.276	1.220	0.175	1.201	0.218	0.947	0.201	0.894
		stdev	85675.009	29.093	0.688	0.171	1.076	0.125	0.789	0.152	0.506	0.162	0.619	0.162	0.537	0.141
		max	550749.000	194.516	3.168	1.607	5.305	1.607	4.019	1.638	2.590	1.841	2.392	1.310	2.590	1.216
	0.50	min	136901.980	47.363	0.480	1.216	0.547	1.197	0.528	1.167	0.391	1.206	0.463	1.027	0.325	1.008
		stdev	252315.982	85.997	1.093	0.177	1.020	0.140	1.136	0.177	1.066	0.133	0.967	0.149	0.775	0.124
		max	1557999.000	522.491	4.211	1.576	3.880	1.528	4.617	1.591	4.348	1.498	3.656	1.497	3.656	1.311
	0.75	min	114468.160	45.881	0.323	1.086	0.390	1.110	0.879	1.054	1.495	1.068	0.261	1.395	0.142	1.424
		stdev	129053.856	51.818	0.840	0.136	1.026	0.149	2.063	0.162	2.800	0.170	0.781	0.254	0.464	0.251
		max	741702.000	294.825	3.246	1.342	4.942	1.404	9.163	1.404	9.411	1.420	3.246	2.138	2.415	1.840
14	0.25	min	130813.680	43.966	0.163	1.968	0.178	1.914	0.136	1.944	0.090	1.883	0.047	1.489	0.122	1.335
		stdev	327876.516	118.075	0.364	0.333	0.430	0.253	0.353	0.334	0.314	0.315	0.138	0.268	0.367	0.254
		max	1731456.000	652.112	1.704	2.886	1.947	2.637	1.752	2.995	1.704	2.964	0.626	2.059	1.572	1.857
	0.50	min	2265773.300	791.922	0.403	1.842	0.308	1.899	0.535	1.841	0.845	1.831	0.349	1.621	0.208	1.583
		stdev	7714071.195	2558.963	0.800	0.222	0.640	0.234	1.130	0.226	1.874	0.246	0.732	0.236	0.410	0.267
		max	53542762.000	17535.788	3.997	2.262	2.559	2.559	5.272	2.278	8.529	2.449	2.786	2.152	1.746	2.231
	0.75	min	2098528.640	924.641	0.484	1.711	0.558	1.782	0.930	1.761	0.917	1.736	0.784	2.696	0.516	2.727
		stdev	4105715.259	1804.679	1.090	0.257	1.293	0.273	1.660	0.297	1.634	0.281	1.552	0.688	1.235	0.761
		max	24691450.000	10719.761	4.472	2.340	4.938	2.496	6.515	2.465	5.890	2.262	5.823	4.259	4.938	4.492
16	0.25	min	770818.680	264.906	0.190	2.642	0.097	2.646	0.145	2.609	0.111	2.640	0.150	2.064	0.098	1.923
		stdev	1480787.395	507.379	0.514	0.409	0.216	0.392	0.340	0.379	0.282	0.485	0.285	0.438	0.227	0.391
		max	8438177.000	2726.807	2.874	3.400	1.031	3.900	1.565	3.620	1.631	4.134	1.254	3.260	1.097	2.840
	0.50	min	42578727.660	16070.926	0.466	2.497	0.271	2.556	0.376	2.421	0.427	2.521	0.313	2.355	0.356	2.319
		stdev	94271771.587	34519.158	0.743	0.361	0.570	0.381	0.700	0.372	0.793	0.354	0.733	0.339	0.644	0.335
		max	575437879.000	207045.670	2.506	3.291	2.505	3.416	3.339	3.603	3.356	3.697	3.522	3.214	2.506	2.964
	0.75	min	91528074.240	45095.698	0.195	2.231	0.276	2.313	1.237	2.095	0.925	2.244	0.478	5.405	0.490	5.400
		stdev	115878982.281	57796.367	0.461	0.383	0.505	0.287	2.409	0.440	1.904	0.404	1.090	1.576	1.119	1.570
		max	467692412.000	243136.801	2.434	2.979	1.527	2.855	8.808	3.759	8.006	3.104	5.827	9.765	5.827	9.080

was less than 5%. In terms of computation time, the 3~9 s CPU time of the GAs is favorable over that of B&B in environments where the scheduling problem has tight time constraints.



<Figure 1> Average of the Mean RDPs for GAs Using Different Initial Population Methods

A comparative analysis of the performance of each GA was conducted by computing the RDPs among the GAs by Eq. 16 and comparing the mean, standard deviation, and maximum values. The results are given in <Table 2>. In terms of the initial population, the method of random generation gave better RDP values, indicating that special methods for generating the initial population did not affect the quality of the final solution. The initial population only influenced the convergence of the GA when agent B had a tight upper bound U. <Figure 1> shows the average of the mean RDPs for GAs using three initial population methods and different combinations of (n, α), where $GA(IP_k)$, $k=1, 2, 3$, represents GAs using IP_k as the initial population method. Note that GAs using IP_3 had low average RDPs in eight of the twelve configurations. Even when this does not give

<Table 2> Relative Deviation Percentage of GAs

n	α	Value of	BB	$GA(IP_1, CO_1)$	$GA(IP_1, CO_2)$	$GA(IP_2, CO_1)$	$GA(IP_2, CO_2)$	$GA(IP_3, CO_1)$
10	0.25	min	0.096	0.113	0.137	0.192	0.038	0.059
		stdev	0.291	0.302	0.398	0.591	0.147	0.264
		max	1.328	1.652	1.986	3.292	0.860	1.627
	0.50	min	0.635	0.460	0.750	0.556	0.857	0.688
		stdev	1.929	1.873	2.309	1.984	2.548	2.604
		max	11.616	11.820	11.616	11.616	11.820	11.820
	0.75	min	1.183	1.316	2.798	2.747	1.468	0.429
		stdev	5.150	4.995	6.344	6.477	5.613	2.335
		max	32.584	31.522	32.584	32.584	31.522	15.528
12	0.25	min	0.530	0.697	0.535	0.356	0.371	0.528
		stdev	1.626	1.812	1.469	1.123	1.087	1.758
		max	8.906	8.906	6.640	6.068	6.068	8.906
	0.50	min	1.045	0.949	0.733	0.530	0.482	0.412
		stdev	3.707	3.415	1.920	1.675	1.140	1.327
		max	23.466	23.466	9.671	9.671	4.617	7.710
	0.75	min	0.604	0.829	2.020	2.965	0.554	0.606
		stdev	1.836	2.363	6.007	6.904	1.828	2.125
		max	9.411	12.316	36.354	36.354	9.411	12.316
14	0.25	min	0.214	0.230	0.347	0.173	0.153	0.192
		stdev	0.561	0.623	1.617	0.565	0.610	0.527
		max	3.318	3.366	11.325	3.144	3.845	2.331
	0.50	min	0.598	0.495	0.893	1.352	0.458	0.327
		stdev	1.381	1.241	1.988	2.798	1.052	0.928
		max	5.631	6.581	8.603	11.910	4.377	4.377
	0.75	min	0.629	0.766	1.194	1.600	1.172	0.838
		stdev	2.138	2.449	2.812	3.585	2.840	2.480
		max	12.954	13.258	13.258	19.931	12.954	13.258
16	0.25	min	0.261	0.096	0.188	0.159	0.176	0.123
		stdev	0.768	0.237	0.485	0.563	0.416	0.399
		max	3.677	1.031	2.795	3.612	2.469	2.469
	0.50	min	0.698	0.892	0.594	0.851	0.572	0.477
		stdev	1.789	4.135	1.521	2.470	1.855	1.176
		max	10.153	28.580	8.210	14.576	10.153	6.913
	0.75	min	0.836	0.582	2.195	1.437	0.907	0.861
		stdev	2.826	1.950	4.294	3.127	2.353	2.283
		max	13.102	13.102	21.386	13.062	13.062	13.062

the best result, the difference from the best is less than 0.5%. Comparing two GAs using IP_3, $GA(IP_3, CO_2)$ is superior to $GA(IP_3, CO_1)$ because it gave a low average RDP in more cases. Based on these observations, we claim that the GA using a random initial population and the two-point crossover operation, represented by $GA(IP_3, CO_2)$, is superior to the other GAs.

The number of generated nodes and CPU time of B&B increased exponentially with the number of jobs n, whereas the CPU time of the GAs increased linearly within the range 0.5~5.5 s. For the largest configuration of $(n, \alpha) = (16, 0.75)$, B&B took an average of 45,095.698 s (12.53h) to find an optimal solution, whereas the GAs took only 2~5 s, depending on the method of generating the initial population. More precisely, GAs using IP_1 and IP_2 for the initial population took about 2 s, while GAs using IP_3 (the random initial population) required around 5.4 s.

It would also appear that the CPU times were affected by the value of and the method of generating the initial population. For a fixed value of n, small values of α (i.e., 0.25 and 0.50) result in faster convergence for those GAs using IP_3. However, GAs using IP_3 with $\alpha = 0.75$ were slower than others. Therefore, it seems that the exploration ability of GA is restricted by the tightness of U, and a specific method for generating the initial population allows the GA to converge faster than when the initial population is generated randomly. However, fast convergence does not necessarily mean that the solution is of good quality.

5. Conclusions

In this paper, we considered the two-agent single-machine scheduling problem with linear job-dependent position-based learning effects, where each job has a different learning ratio. The objective was to minimize the total weighted completion time for one agent, with the restriction that the makespan of the other agent cannot exceed a given upper bound. Since this problem is at least binary NP-hard, we suggested some efficient GAs using different methods for generating the initial population and crossover operations. A B&B algorithm incorporating several dominance properties and a lower bound was developed to find the optimal solution. The computational results indicate that the B&B algorithm could solve instances with up to 14 jobs in a reasonable amount of CPU time, and it was found that $GA(IP_3, CO_2)$, which uses a random initial population and the two-point crossover operation, performed well in almost every configuration. Overall, from the perspective of computation time, the 3~9 s of CPU time required by the GAs will generally be preferable to the CPU time of the B&B approach in environments where the scheduling problem has tight time constraints.

In future research, we will extend the current method in three ways. First, we will consider other performance objectives for the two agents, such as minimizing tardiness, weighted tardiness, or the number of tardy jobs. These are practical issues in industry, although they can make the problem more difficult. Second, we will use other learning effects so that the actual processing time is dependent on the sum of processing times of preceding jobs. This learning effect is nonlinear, making it more difficult to find optimal solutions to the two-agent scheduling problem. Finally, extensions to the multi-agent or multi-machine environments are another interesting topic for consideration.

References

[1] Agnetis, A., Mirchandani, P.B., Pacciarelli, D., and Pacifici, A., Scheduling problems with two competing agents. *Operations Research*, 2004, Vol. 52, No. 2, pp. 229-242.

[2] Agnetis, A., Pacciarelli, D., and Pacifici A., Multi-agent single machine scheduling. *Annals of Operations Research*, 2007, Vol. 150, No. 1, pp. 3-15.

[3] Bachman, A. and Janiak, A., Scheduling jobs with position-dependent processing times. *Journal of the Operational Research Society*, 2004, Vol. 55, pp. 257-264.

[4] Baker, K.R. and Smith, J.C., A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 2003, Vol. 6, No. 1, pp. 7-16.

[5] Biskup D., Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 1999, Vol. 115, No. 1, pp. 173-178.

[6] Biskup, D., A state-of-the-art review on scheduling with learning considerations. *European Journal of Operational Research*, 2008, Vol. 188, No. 2, pp. 315-329.

[7] Chang, P.C., Chen, S.H., and Mani, V., A note on due-date assignment and single machine scheduling with a learning and aging effect. *International Journal of Production Economics*, 2009, Vol. 117, No. 1, pp. 142-149.

[8] Cheng, T.C.E. and Wang, G., Single machine scheduling with learning effect considerations. *Annals of Operations Research*, 2000, Vol. 98, No. 1, pp. 273-290.

[9] Cheng, T.C.E., Ng, C.T., and Yuna, J.J., Multi-agent

scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, 2006, Vol. 362, No. 1-3, pp. 273-281.

[10] Cheng, T.C.E., Wu, W.H., Cheng, S.R., and Wu, C.C., Two-agent scheduling with position-based deterioration jobs and learning effects. *Applied Mathematics and Computation*, 2011, Vol. 217, No. 1, pp. 8804-8824.

[11] Ding, G. and Sun, S., Single-machine scheduling problems with two agents competing for makespan. *Life System Modeling and Intelligent Computing*, 2010, Vol. 6328, pp. 244-255.

[12] Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy, Kan AHG., Optimization and approximation in deterministic sequencing and scheduling theory : a survey. *Annals of Discrete Mathematics*, 1979, Vol. 5, pp. 287-326.

[13] Hardy, G., Littlewood, J., and Polya, G. Inequalities. London : Cambridge University Press, 1967.

[14] Hillier, F.S. and Lieberman, G.J., *Introduction to Operations Research*, McGraw Hill, 2015.

[15] Knotts, G., Dror, M., and Hartman, B.C., Agent-based project scheduling. *IIE Transactions*, 2000, Vol. 32, No. 5, pp. 387-401.

[16] Kuo, W.H. and Yang, D.L., Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *Journal of the Operational Research Society*, 2008, Vol. 59, pp. 416-420.

[17] Lee, W.C., Wang, W.J., Shiau, Y.R., and Wu, C.C., A single-machine scheduling problem with two-agent and deteriorating jobs. *Applied Mathematical Modelling*, 2010, Vol. 34, No. 10, pp. 3098-3107.

[18] Leung, J.Y.T., Pinedo, M.L., and Wan, G., Competitive two-agent scheduling and its applications *Operations Research*, 2010, Vol. 58, No. 2, pp. 458-469.

[19] Li, D.C. and Hsu, P.H., Solving a two-agent single-machine scheduling problem considering learning effect. *Computers and Operations Research,* 2012, Vol. 39, No. 7, pp. 1644-1651.

[20] Liu, P., Yi, N., and Zhou, X., Two-agent single-machine scheduling problems under increasing linear deterioration. *Applied Mathematical Modelling*, 2011, Vol. 35, No. 5, pp. 2290-2296.

[21] Liu, P., Zhou, X., and Tang, L., Two-agent single-machine scheduling with position-dependent processing times. *International Journal of Advanced Manufacturing Technology*, 2010, Vol. 48, No. 1, pp. 325-331.

[22] Mitchell, M., An introduction to genetic algorithm, MIT Press, 1996.

[23] Mosheiov, G., A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling*, 2005, Vol. 41, No. 8-9, pp. 883-886.

[24] Mosheiov, G., Parallel machine scheduling with a learning effect. *Journal of the Operational Research Society*, 2001, Vol. 52, No. 10, pp. 1165-1169.

[25] Ng, C.T., Cheng, T.C.E., and Yuan, J.J., A note on the complexity of the problem of two-agent scheduling on a single machine. *Journal of Combinatorial Optimization*, 2006, Vol. 12, No. 4, pp. 387-394.

[26] Pessan, C., Bouquard, J.L., and Neron, E., An unrelated parallel machines model for an industrial production resetting problem. *European Journal of Industrial Engineering*, 2008, Vol. 2, No. 2, pp. 153-171.

[27] Wang, J. and Wang, M., Worst-case analysis for flow shop scheduling problems with an exponential learning effect. *Journal of the Operational Research Society*, 2012, Vol. 63, pp. 130-137.

[28] Wang, J.B. and Xia, Z.Q., Flow-shop scheduling with a learning effect. *Journal of the Operational Research Society*, 2005, Vol. 56, pp. 1325-1330.

[29] Wu, C.C., Huang, S.K., and Lee, W.C., Two-agent scheduling with learning consideration. *Computers and Industrial Engineering*, 2011b, Vol. 61, No. 4, pp. 1324-1335.

[30] Wu, C.C., Yin, Y., and Cheng, S.R., Some single-machine scheduling problems with a truncation learning effect. *Computers and Industrial Engineering*, 2011a, Vol. 60, No. 4, pp. 790-795.

[31] Wu, W.H., Xu, J., Wu, W.H., Yin, Y., Cheng, I.F., and Wu, C.C., A tabu method for a two-agent single-machine scheduling with deterioration jobs. *Computers and Operations Research*, 2013, Vol. 40, No. 8, pp. 2116-2127.

[32] Yin, Y., Cheng, S.R., Cheng, T.C.E., Wu, W.H., and Wu, C.C., Two-agent single-machine scheduling with release times and deadlines. *International Journal of Shipping and Transport Logistics*, 2013, Vol. 5, No. 1, pp. 75-94.

[33] Yin, Y., Xu, D., Cheng, S.R., and Wu, C.C., A generalization model of learning and deteriorating effects on a single-machine scheduling with past-sequence-dependent setup times. *International Journal of Computer Integrated Manufacturing*, 2012, Vol. 25, No. 9, pp. 804-813.

ORCID

Jin Young Choi | http://orcid.org/0000-0001-6397-3107