한국시뮬레이션학회 논문지
Vol. 24, No. 4, pp. 77-88 (2015. 12)
http://dx.doi.org/10.9709/JKSS.2015.24.4.077
ISSN 1225-5904

Simulated Annealing for Two-Agent Scheduling Problem with Exponential Job-Dependent Position-Based Learning Effects

Jin Young Choi*

작업별 위치기반 지수학습 효과를 갖는 2-에이전트 스케줄링 문제를 위한 시뮬레이티드 어닐링

최진영*

ABSTRACT

In this paper, we consider a two-agent single-machine scheduling problem with exponential job-dependent position-based learning effects. The objective is to minimize the total weighted completion time of one agent with the restriction that the makespan of the other agent cannot exceed an upper bound. First, we propose a branch-and-bound algorithm by developing some dominance /feasibility properties and a lower bound to find an optimal solution. Second, we design an efficient simulated annealing (SA) algorithm to search a near optimal solution by considering six different SAs to generate initial solutions. We show the performance superiority of the suggested SA using a numerical experiment. Specifically, we verify that there is no significant difference in the performance of $\%errors$ between different considered SAs using the paired t-test. Furthermore, we testify that random generation method is better than the others for agent A, whereas the initial solution method for agent B did not affect the performance of $\%error$.

Key words : Two-Agent Scheduling, Exponential Learning Effect, Job-Dependent Position-Based Processing Time, Total Weighted Completion Time, Makespan, Branch-And-Bound Algorithm, Simulated Annealing

요 약

본 논문은 작업별 위치기반 지수학습 효과를 갖는 2-에이전트 단일기계 스케줄링 문제를 고려한다. 에이전트 A는 가중 완료 시간의 합을 최소화하며, 에이전트 B는 총소요시간에 대한 상한 값을 만족하는 조건을 갖는다. 본 연구에서는 먼저 우수해/가능해에 대한 특성을 개발하고, 이를 이용하여 최적 해를 찾기 위한 분지한계 알고리즘을 설계한다. 또한 근사 최적 해를 구하기 위해 6가지 다른 초기해 생성 방법을 이용한 시뮬레이티드 어닐링 알고리즘을 제안한다. 수치 실험을 통해 제안된 알고리즘의 우수한 성능을 검증한다. 실험 결과, 다른 초기해 생성 방법들 간에는 $\%errors$ 차이가 유의하게 발생하지 않았으며, 에이전트 A의 작업 순서를 무작위로 생성할 때 성능이 좋아짐을 발견하였다. 반면에, 에이전트 B의 초기해 생성 방법은 성능에 영향을 미치지 않았다.

주요어 : 2-에이전트 스케줄링, 지수 학습효과, 작업별 위치기반 처리시간, 가중 완료 시간의 합, 총소요시간, 분지 한계 알고리즘, 시뮬레이티드 어닐링

1. Introduction

Two-agent single-machine scheduling problem can be found in various industrial applications where two agents compete for a single common machine to achieve their respective objectives. For example, we can consider a

Received: 28 August 2015, **Revised:** 8 December 2015, **Accepted:** 10 December 2015

***Corresponding Author:** Jin Young Choi
E-mail: choijy@ajou.ac.kr

An abridged version of this article was presented at the MISTA 2015 Conference (Choi, 2015).

production system under maintenance planning. The production department (agent) wants to operate the system without any idle time in order to maximize the system utilization. On the other hand, the maintenance department (agent) calls for frequent pauses of the production system in order to reduce the number of unexpected breakdowns of the system. Therefore, two departments (agents) share the production system, while pursuing different objective functions. Another example of the two-agent scheduling problems is the case of a classical problem in air traffic management for scheduling aircraft landings on a given set of runways. If two airlines (agents) have different performance measures such as safety and quality of service, each airline (agent) is definitely interested in maximizing the frequency cleaning/ checking the runways and minimizing the delay of the corresponding flights, respectively, in order to maximize the satisfaction of its own passengers (Agnetis, 2012).

This problem is a special case of general single-agent bi-criteria optimization model (Agnetis et al., 2004) because all jobs contribute to both objectives, and a key issue is to determine the non-dominated schedules for each objective in a sense that a better schedule for one objective makes the other objective worse. Furthermore, this problem has some characteristics as follows: (i) two objectives correspond to different two agents having their own jobs to process, (ii) they compete on the usage of a common processing resource, (iii) each agent has different performance measure, and (iv) only the jobs pertaining to one agent contribute to the computation of the objective function for itself. Therefore, a typical methodology for simple single-agent bi-criteria optimization models such as using weighted objective functions of two agents cannot be applied, and the complexity of it is higher than that by simple single-agent bi-criteria scheduling problem (Agnetis et al., 2004), necessitating a new systematic approach.

Of particular interest is a two-agent single-machine scheduling problem with exponential job-dependent position-based learning effect. This means that each job has its own learning effect, implying that the learning in the production process of some jobs is faster than those of others. Moreover, the actual processing time of a job is expressed as an exponential decreasing function of learning effect and processing sequence. For example, performing similar tasks repeatedly can improve the skills of workers so that they can perform setups and handle raw materials faster, while reducing the actual processing time. This modeling concept is a plausible scenario in real-life manufacturing environment, deserving some attention.

However, to the best of our knowledge, in the literature there has been no research works on this specific issue considering two-agent, position-dependent, and job-dependent learning effect, simultaneously, whereas there were some works on simpler cases considering position-dependent processing times (learning/ aging) for two-agent or job-dependent learning effect for single agent as follows; Since the concepts of the two-agent single-machine scheduling problem was first introduced by Baker and Smith (2003), many research works have been conducted on this topic, while learning/aging effect in a single-agent scheduling problem were firstly introduced by Biskup (1999) and Mosheiov (2001), respectively. Readers can find an extensive survey on scheduling problems with learning effects in Biskup (2008). Mosheiov (2005) showed that a V-shaped schedule is optimal for the problem of minimizing flowtime in a single-machine case. Kuo and Yang (2008) solved a single-machine scheduling problem with the cyclic process of an aging effect. Chang et al. (2009) worked on single-machine scheduling problems with a common due date under learning/aging effect consideration.

Meanwhile, two-agent scheduling problems with position-dependent processing times have received considerable attention in recent years, where actual processing times can be represented by using a linear function or an exponential function. Liu et al. (2010), Lee et al. (2010), Liu et al. (2011), and Wu et al. (2013) considered a linear function for computing actual processing times. On the other hand, Cheng et al. (2011), Wu et al. (2011b), and Li and Hsu (2012) applied an exponential function to represent actual processing times. All of these works took into account different performance objectives for two agents, which are functions of completion times such as (weighted) sum of tardiness, (weighted) sum of completion times, lateness, the number of tardy jobs, or upper bound of makespan, and so on.

Regarding the modeling concept of job-dependent learning

effect, it was suggested by Cheng and Wang (2000) for the first time. Mosheiov and Sidney (2003) studied a job-dependent learning curve where the learning of some jobs is faster than those of others. Then, Bachman and Janiak (2004) investigated a learning effect formulation on the single-machine case, and Wang and Xia (2005) extended it to multiple-machine case for a flow-shop consisting of an increasing series of dominating machines. However, they did not consider a two-agent case competing for a common resource.

Motivated by these remarks, in this paper, we investigate a rather general problem, where two agents are included with an exponential position-based and job-dependent learning effect, while competing for a common single machine. In the context of the operational framework, we call it a two-agent single-machine scheduling problem with exponential job-dependent position-based learning effect. The objective is to minimize the total weighted completion time of one agent with the restriction that the makespan of the other agent cannot exceed an upper bound. We suggest some dominance and feasibility properties for a branch-and-bound algorithm (B&B), which can be used to find optimal solutions and to compare effectiveness of other algorithms considered. Furthermore, we design an efficient simulated annealing (SA) algorithm to search a near optimal solution and show its superiority of performance by using a numerical experiment.

The remainder of this paper is organized as follows. In the next section, we define the problem formally and suggest several dominance and feasibility properties related to the B&B algorithm. In Section 3, we design an efficient simulated annealing algorithm to obtain near-optimal solutions. The computational experiments are conducted in Section 4 and we conclude our discussion by suggesting some future works in Section 5.

2. Problem definition and a branch-and-bound algorithm

2.1 Problem definition

Two agents A and B have sets of jobs $J^A=\left\{J_1^A, J_2^A, \cdots, J_{n_A}^A\right\}$, $J^B=\left\{J_1^B, J_2^B, \cdots, J_{n_B}^B\right\}$ to process, respectively, while competing for a single common machine. The objective of agent A is to minimize the total weighted completion time and agent B wants to keep the makespan, $C_{\max}^B$, less than an upper bound U. Each job for agent A is assigned with a weight w_i^A and a normal processing time $p_i^A, 1 \le i \le n_A$. Each job for agent B has a normal processing time $p_j^B, 1 \le j \le n_B$. All jobs have job-dependent and position-based learning effect, so that actual processing time of job $J_i^X, X \in \{A, B\}$ processed at the rth position in a sequence, $p_i^X(r)$, can be expressed as an exponential decreasing function $p_i^X(r) = p_i^X \bullet r^{-b_i^X}$, where $b_i^X > 0$ is a learning ratio of job $J_i^X, X \in \{A, B\}$. Then, using the three-field notation $\Psi_1|\Psi_2|\Psi_3$ suggested by Graham et al. (1979), where Ψ_1 is the number of machines, Ψ_2 denotes job characteristics, and Ψ_3 describes objective functions, the scheduling problem under consideration can be represented as

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

where C_i^A is the completion time of job J_i^A. Agnetis et al. (2004) showed that $1||\sum_{i=1}^{n_A} w_i^A C_i^A : C_{\max}^B \le U$ is binary NP-hard. It implies that our problem in Eq. 1 is at least binary NP-hard, and we need an efficient solution approach to solve it. In our work, we suggest a B&B algorithm to obtain an optimal solution, and a simulated annealing algorithm for a near optimal solution.

2.2 Properties and a lower bound for a branch-and-bound algorithm

For a B&B approach, we first develop four dominance properties based on a pairwise interchange comparison method as follows. Suppose that we have two schedules S_1 and S_2 s.t. $S_1 = (\pi, J_i^X, J_j^X, \pi')$ and $S_1 = (\pi, J_j^X, J_i^X, \pi')$, where π is a scheduled part of $(r-1)$ jobs and π' is a unscheduled part of $(n-r-1)$ jobs. Hence, jobs J_i^X and J_j^X are in the rth and $(r+1)$th positions in S_1, respectively. S_2 can be obtained by interchanging two jobs J_i^X and J_j^X

in S_1. By defining t as the completion time of the last job in π, we can compute the completion times of jobs J_i^X and J_j^X as

$$C_i(S_1)=t+p_i^X r^{-b_i^X},\ C_j(S_1)=t+p_i^X r^{-b_i^X}+p_j^X(r+1)^{-b_j^X}$$
$$C_j(S_2)=t+p_j^X r^{-b_j^X},\ C_i(S_2)=t+p_j^X r^{-b_j^X}+p_i^X(r+1)^{-b_i^X}.$$

Let us assume that we just sequenced J_j^X after π and are about to arrange J_i^X, resulting in schedule S_2. Then, we can derive certain conditions under which schedule S_2 is dominated by schedule S_1. Specifically, we need two conditions for two agents such that (i) S_1 has smaller total weighted completion time for agent A than that of S_2, and (ii) the makespan of agent B satisfies the upper bound condition. In addition, we need one more condition such that $C_j^X(S_1) < C_i^X(S_2)$, implying that we can keep the dominance of S_1 over S_2 after arranging jobs in π' later steps. By defining $\delta_{ij}^{XX}=b_i^X-b_j^X$, we have the following properties.

(Property 1) For $J_i^X\in J^A, J_j^X\in J^A$,
if $\dfrac{w_j^A}{w_i^A}(r+1)^{\delta_{ij}^{AA}} \le \dfrac{p_i^A}{p_j^A} < \min\left\{1, \dfrac{(r+1)^{b_j^A}-r^{b_j^A}}{(r+1)^{b_i^A}-r^{b_i^A}}\right\} r^{\delta_{ij}^{AA}}$,
then S_1 dominates S_2.

Proof: Since $J_i^X, J_j^X\in J^A$, the first condition is

$$w_i^A C_i^A(S_1)+w_j^A C_j^A(S_1) \le w_j^A(S_2)+w_i^A C_i^A(S_2)$$
$$\Leftrightarrow \left(w_i^A+w_j^A\right)\left(p_i^A r^{-b_i^A}-p_j^A r^{-b_j^A}\right) + \left[w_j^A p_j^A(r+1)^{-b_j^A}-w_i^A p_i^A(r+1)^{-b_i^A}\right] \le 0.$$

Therefore, we have

$$p_i^A r^{-b_i^A}-p_j^A r^{-b_j^A} \le 0 \Leftrightarrow \frac{p_i^A}{p_j^A} \le r^{\delta_{ij}^{AA}} \quad (2)$$

$$w_j^A p_j^A(r+1)^{-b_j^A}-w_i^A p_i^A(r+1)^{-b_i^A} \le 0 \quad (3)$$
$$\Leftrightarrow \frac{p_i^A}{p_j^A} \ge \frac{w_j^A}{w_i^A}(r+1)^{\delta_{ij}^{AA}}.$$

From $C_j^A(S_1) < C_i^A(S_2)$, we have

$$p_i^A\left[r^{-b_i^A}-(r+1)^{-b_i^A}\right] < p_j^A\left[r^{-b_j^A}-(r+1)^{-b_j^A}\right] \quad (4)$$
$$\Leftrightarrow \frac{p_i^A}{p_j^A} < \frac{\left[(r+1)^{b_j^A}-r^{b_j^A}\right][r(r+1)]^{\delta_{ij}^{AA}}}{(r+1)^{b_i^A}-r^{b_i^A}}.$$

Since there is no job for agent B, we do not need the makespan condition for agent B. From Eqs. 2-4, S_1 dominates S_2, if $\dfrac{w_j^A}{w_i^A}(r+1)^{\delta_{ij}^{AA}} \le \dfrac{p_i^A}{p_j^A} < \min\left\{1, \dfrac{(r+1)^{b_j^A}-r^{b_j^A}}{(r+1)^{b_i^A}-r^{b_i^A}}\right\} r^{\delta_{ij}^{AA}}$.

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

(Property 2) For $J_i^X\in J^A, J_j^X\in J^B$,
if $(i)\ \dfrac{p_i^A}{p_j^B} < \dfrac{(r+1)^{b_j^B}-r^{b_j^B}}{(r+1)^{b_i^A}-r^{b_i^A}}[r(r+1)]^{\delta_{ij}^{AB}}$
and $(ii)\ t+p_i^A r^{-b_i^A}+p_j^B(r+1)^{-b_j^B} \le U$,
then S_1 dominates S_2.

(Property 3) For $J_i^X\in J^B, J_j^X\in J^A$,
if $(i)\ \dfrac{p_i^B}{p_j^A} < \left[1-\left(\dfrac{r}{r+1}\right)^{b_j^A}\right] r^{\delta_{ij}^{BA}}$
and $(ii)\ t+p_i^B r^{-b_i^B} \le U$, then S_1 dominates S_2.

(Property 4) $J_i^X\in J^B, J_j^X\in J^B$,
if $\dfrac{w_j^B}{w_i^B}(r+1)^{\delta_{ij}^{BB}} \le \dfrac{p_i^B}{p_j^B} < \min\left\{1, \dfrac{(r+1)^{b_j^B}-r^{b_j^B}}{(r+1)^{b_i^B}-r^{b_i^B}}\right\} r^{\delta_{ij}^{BB}}$,
then S_1 dominates S_2.

Moreover, we have three more feasibility properties of a sequence as follows. Suppose that we have a sequence of jobs $S=(\pi,\pi^c)$, where π represents a sequence of k jobs scheduled and π^c is a set of $(n-k)$ jobs unscheduled. In addition, let $\pi_{[k]}$ and $C_{[k]}$ be the last kth job scheduled and the completion time of it, respectively. Then, the following properties can be proved easily using the objective function requirement of agent B.

(Property 5) If $C_{[k]} < U$
with $\pi_{[k]}\in J^A$ and $\left(\pi^c\cap J^B\right)\neq\varnothing$,
then the sequence (π,π^c) is non-promising.

(Property 6) If $C_{[k]} > U$
with $\pi_{[k]} \in J^B$,
then the sequence (π, π^c) is non-promising.

(Property 7) If $C_{[k]} \leq U, (\pi^c \cap J^B) \neq \varnothing$,
and $C_{[k]} + p^{\min}(k+1) > U$,
then the sequence (π, π^c) is non-promising, where $p^{\min}(k+1)$ is the minimum actual processing time by any job in π^c at the $(k+1)$th position.

Based on these properties, we can apply a B&B algorithm to find an optimal solution (Hillier and Lieberman, 2015). A node represents a partial or complete schedule and the initial node is empty, denoting none of jobs is scheduled. Then, we can assign any job in the first position in a sequence and select a job for the second position, and so on. This is called a branching process. Hence, the basic idea is to branch a node into several nodes, each corresponding to scheduling one available job at that time point, and bound it by computing the potential minimum value of the total weighted completion time, called a lower bound, for agent A, which can speed up the search procedure.

(Lemma 1) Suppose that there are $n_2 > 0$ jobs for agent A in π^c so that their weights can be sorted in a non-increasing order, as $w^A_{(1)} \geq w^A_{(2)} \geq \cdots \geq w^A_{(n_2)}$. Then, a lower bound of $\sum_{i=1}^{n_A} w_i^A C_i^A(S)$ for a sequence $S = (\pi, \pi^c)$ corresponding to the state assigning only jobs in π is

$$LB(S) = \sum_{r=1}^{n_1} w^A_{[r]} C^A_{[r]} + \sum_{r=1}^{n_2} w^A_{(r)} C^A_{(r)} \quad (5)$$

where $n_1 = n_A - n_2$, and $w^A_{[r]}$ and $C^A_{[r]}$ denote the weight and the completion time of the job scheduled in the rth order among n_1 jobs for agent A in π, respectively. Furthermore, $C^A_{(r)}$ denotes the estimated minimum completion time of a job that can be scheduled in the rth order among n_2 jobs for agent A in π^c.

Proof: The first term in Eq. 5 computes the sum of weighted completion times of jobs for agent A already scheduled. In the second term, without loss of generality, completion times, $C^A_{(r)}$, are an increasing function of the processing sequence, and $w^A_{(r)}$s' are decreasing function. Therefore, $\sum_{r=1}^{n_2} w^A_{(r)} C^A_{(r)}$ is minimized (Hardy et al., 1967), providing a lower bound for S.

For a given schedule S, we suggest an efficient method to calculate $C^A_{(r)}$ in Eq. 5 as follows.

1. Identify the minimum actual processing times at the positions from $(k+1)$th to nth in S, while allowing to select them from same jobs in π^c.
2. Calculate completion times at those positions using the minimum actual processing times, denoted by $\hat{C}_{[l]}, l = k+1, \cdots, n$.
3. Assign n_2 jobs for agent A to first n_2 positions and $(n-k-n_2)$ jobs for agent B to following $(n-k-n_2)$ positions.
4. If $n-k-n_2 > 0$, adjust the sequence of $(n-k)$ jobs to satisfy the upper bound condition for agent B as follows. Otherwise, go to (5).
   (a) Check the upper bound condition of agent B's jobs from the nth job in a backward manner.
   (b) Whenever a job does not satisfy the upper bound condition, find one preceding last job for agent A and interchange with it.
   (c) Update U as $U - \hat{p^A}$, where $\hat{p^A}$ is the minimum actual processing time of the selected agent A's job for interchanging.
   (d) Repeat (b) and (c) until we find first agent B's job satisfying the upper bound condition, or there is no more jobs to consider.
5. Stop with the minimum completion times of n_2 jobs for agent A.

3. Design of SA using different initial solutions

As an efficient solution approach to find a near optimal solution, in this section, we suggest a simulated annealing (SA) algorithm, that was proposed by Kirkpatrick et al.

(1983). It is one of the most popular meta-heuristics used to solve combinatorial optimization problems based on trajectory search procedure. It starts from an initial trial solution and explores the solution space by taking steps in random direction, while accepting some deteriorating steps probabilistically. Therefore, it can escape from a local optimum and increase the possibility to find a better solution. As the algorithm proceeds, it focuses on the feasible region that might contain an optimal solution. The main features of the algorithm can be described as follows.

Objective function : During the search procedure, we evaluate a trial solution under consideration using the objective function for agent A, which is to minimize the total weighted completion time of jobs for agent A. We represent the objective function value for the current trial solution and next trial solution as z_c and z_n, respectively.

Initial solution : Since we need to generate a feasible initial solution and the upper bound condition for agent B affects the feasibility of a solution under consideration, we first arrange the jobs for agent B ahead in generating an initial solution, to make $C_{\max}^{B}$ as small as possible, following the jobs for agent A. We can consider different methods to make the partial sequences for the two agents. Specifically, we consider three methods to arrange jobs for agent A such as (i) IS_1^A = random order, (ii) IS_2^A = shortest normal processing time (SPT) order, and (iii) IS_3^A = shortest weighted normal processing time (WSPT) order. This is because the objective of agent A is to minimize the total weighted completion time, and it might be achieved by arranging jobs based on the information of (weighted) normal processing times. In the case of scheduling jobs for agent B, we suggest two methods to order jobs for agent B such as (i) IS_1^B = random order and (ii) IS_2^B = non-decreasing order of b_j^B. Therefore, we can consider $3\times 2=6$ different methods to generate an initial solution.

Neighborhood generation : For a given trial solution, we select two jobs randomly and interchange them to generate next trial solution. After computing $C_{\max}^{B}$ of new sequence generated, we check the upper bound condition for agent B. If it is feasible, we call it next trial solution. Otherwise, we reapply this procedure until we get a feasible one.

Move selection : Using the next trial solution, we compute the objective function and get z_n. If $z_c > z_n$, we accept it and update the current trial solution with it. Otherwise, we can accept it with the acceptance probability defined as

$$P_a = e^{\frac{z_c - z_n}{T}} \tag{6}$$

where T is a control parameter, called the temperature, which can change the tendency to accept a worse solution than the current one. If $z_c < z_n$, the exponent part of Eq. 6 becomes negative, making P_a in the range 0-1 as the probability. If it is rejected, then we repeat from the neighborhood generation step.

Temperature schedule : We notice that we can change the value of T to control the acceptance probability. Since P_a becomes large as the value of T is large, we can make the search of SA to perform in random direction by setting a large value of T at the early stage, whereas we can focus on a special region by using small value of T at the later stage of the algorithm. Based on this idea, we can design how to change the temperature, called the temperature schedule. First, we set the initial temperature using the initial objective function value as $T_1 = c_1 \times z_c \ (0 < c_1 < 1)$. Then, after performing a fixed number of iterations N at T_1, we decrease the value of T_1 by $(1-c_2)\times 100\%$, represented as $T_2 = c_2 \times T_1 \ (0 < c_2 < 1)$. We can repeat this procedure for a fixed number of steps C, which is the number of times we change the temperature. Hence, the temperature schedule can be expressed as $T_y = c_2 \times T_{y-1}$, $y = 2, 3, \cdots, C$.

Terminating condition : We terminate the algorithm after a fixed number of iterations defined as $C\times N$.

4. A numerical experiment

4.1 Design of a numerical experiment

We designed a numerical experiment to evaluate the performance of the suggested B&B algorithm and six different $SA(IS_h^A, IS_d^B)$ $(h=1,2,3, d=1,2)$, using different methods in the generation of the initial solution as follows. First, we considered four different values of n as $n=10, 12, 14, 16$ with $n_A=n_B$. The value of U was set by $U=(1+\alpha)U_{\min}+\alpha U_{\max}$, where $U_{\min}$ and $U_{\max}$ are the minimum and maximum value of the makespan that can be made using all jobs for agent B, respectively, and $\alpha\,(0<\alpha<1)$ is a real-valued parameter. We considered three different values of α as $\alpha=0.25, 0.50, 0.75$. We computed the value of $U_{\max}$ as the sum of normal processing times of jobs for agent B and $U_{\min}$ as the sum of minimum actual processing times at first n_B positions in a sequence by allowing duplicated selection of jobs.

We performed a pre-testing of SA by generating some simple scheduling problems using $n=6, 8$ and $\alpha=0.25, 0.50, 0.75$. More precisely, for each configuration of (n,α), we generated 30 problem instances using normal processing times and job weights randomly generated in the range $[1, 100]$, and learning effects randomly generated in the range $(0, 2]$. We solved problem instances using six SAs, and compared the performance of the SAs using $\%\,error$ defined as

$$\%\,error=\left(\frac{TWCT\ by\ SA(IS_h^A, IS_d^B)}{Optimal\ \ TWCT}-1\right)\times 100$$

where $TWCT$ is the total weighted completion time, and $SA(IS_h^A, IS_d^B)$ $(h=1,2,3, d=1,2)$ represents the SA using IS_h^A and IS_d^B as initial solution methods for agent A and agent B, respectively, as defined in Section 3. Using the pre-testing results, we set the parameter values of SA to $c_1=0.2$, $c_2=0.5$, $N=10\times n$, $C=10$.

As a relative comparison method of the performance of different SAs, we defined the relative deviation percentage (RDP) of $SA(IS_h^A, IS_d^B)$ as

$$RDP_{h,d}=\left(\frac{TWCT\ by\ SA(IS_h^A, IS_d^B)}{\min\ TWCT\ by\ any\ SA}-1\right)\times 100.$$

We can calculate this performance measure by computing one RDP for each problem instance and compiling statistics such as mean, maximum, or standard deviation.

4.2 Experimental results

For each configuration of (n,α) and the solution method, we generated 50 problem instances using the same parameter ranges and solved them using the B&B algorithm and six different SAs, $SA(IS_h^A, IS_d^B)$ $(h=1,2,3, d=1,2)$. For each combination of (n,α) and B&B (or $SA(IS_h^A, IS_d^B)$), we calculated the mean, standard deviation (stdev), and maximum (max) number of generated nodes, CPU time (in seconds), and $\%\,error$ as in Table 1. Because small value of α decreases the value of U, it generates a tight upper bound for agent B and makes B&B difficult to find an optimal solution, which were expressed as increasing values of node number and CPU time at $\alpha=0.25$, compared to other values of α, for a fixed value of n. The number of generated nodes and CPU time of B&B increased exponentially as n increases. Specifically, B&B took a mean of 218,321.058 s (60.64 h) to find an optimal solution for the largest system $(n,\alpha)=(16, 0.25)$.

The SAs showed good performance because they had low $\%\,error$ in almost all configurations of (n,α) with a mean of less than 2%. The CPU time is within 1.1 s in all configurations, that is obviously favorable over that of B&B in environments requiring real-time scheduling. The CPU times were not affected by the size n of problem instances because the CPU times were increased linearly as n increases for a fixed value of α. However, we could not see any linear relationship between the CPU times and the value of α for a fixed value of n. This can imply that the tightness of upper bound U for agent B does not affect the computation time of SA procedure.

The performance difference of the proposed SAs in terms of $\%\,error$ was compared statistically by applying the paired t-test. After denoting the difference of the ith paired values of $\%\,error$ for two considered SAs, (for example, $SA(IS_1^A, IS_1^B)$ vs. $SA(IS_1^A, IS_2^B)$), as D_i, we

Table 1. Results of numerical experiments (CPU time in s)

n	α	value of	B&B		$SA(IS_1^A, IS_1^B)$		$SA(IS_1^A, IS_2^B)$		$SA(IS_2^A, IS_1^B)$		$SA(IS_2^A, IS_2^B)$		$SA(IS_3^A, IS_1^B)$		$SA(IS_3^A, IS_2^B)$	
			Nodes	CPU time	% error	CPU time	% error	CPU time	% error	CPU time	% error	CPU time	% error	CPU time	% error	CPU time
10	0.25	mean	6,094,740	3.401	0.132	0.470	0.159	0.456	1.046	0.487	0.888	0.476	0.485	0.459	0.297	0.445
		stdev	11,291,003	6.306	0.466	0.105	0.422	0.067	3.123	0.101	3.083	0.071	1.351	0.053	0.726	0.050
		max	60,306,000	34.616	3.131	1.077	2.470	0.780	13.946	1.014	16.921	0.764	8.037	0.687	3.189	0.640
	0.50	mean	3,543,460	1.902	0.175	0.448	0.519	0.462	0.361	0.477	0.146	0.454	0.480	0.456	0.395	0.442
		stdev	5,846,632	2.865	0.615	0.052	1.569	0.073	1.098	0.053	0.326	0.053	2.578	0.052	1.147	0.059
		max	37,106,000	17.659	3.775	0.592	7.842	0.749	5.456	0.593	1.461	0.609	18.216	0.577	6.904	0.592
	0.75	mean	3,069,100	1.614	0.623	0.414	0.795	0.412	0.296	0.434	0.628	0.413	0.114	0.419	0.330	0.407
		stdev	3,584,598	1.864	1.519	0.024	3.099	0.019	0.724	0.026	2.338	0.024	0.242	0.027	0.862	0.022
		max	15,463,000	7.987	7.411	0.514	20.197	0.484	3.411	0.484	15.669	0.515	1.183	0.561	5.132	0.483
12	0.25	mean	299,060,680	156.498	0.667	0.599	1.329	0.630	1.615	0.610	1.046	0.599	0.735	0.602	2.238	0.591
		stdev	811,833,870	427.702	1.382	0.051	3.764	0.071	4.493	0.047	2.615	0.051	1.685	0.054	5.267	0.064
		max	4,073,493,000	2,113.460	8.975	0.764	21.614	0.764	23.697	0.733	13.842	0.780	10.733	0.795	23.697	0.780
	0.50	mean	57,654,520	29.787	0.622	0.608	0.659	0.642	0.753	0.609	0.880	0.611	0.747	0.603	0.458	0.593
		stdev	126,014,408	65.898	1.034	0.027	1.680	0.030	1.451	0.028	3.171	0.036	1.618	0.019	0.852	0.029
		max	584,804,000	315.667	3.929	0.733	10.315	0.717	5.762	0.702	21.296	0.827	9.747	0.671	3.627	0.671
	0.75	mean	131,029,020	69.099	0.930	0.613	0.734	0.650	0.812	0.609	1.251	0.603	0.729	0.602	0.502	0.575
		stdev	473,526,550	249.851	2.541	0.020	2.355	0.036	2.837	0.023	3.439	0.028	1.546	0.022	1.387	0.025
		max	3,271,777,000	1,727.641	15.057	0.702	15.262	0.842	18.453	0.671	20.679	0.749	8.881	0.671	6.759	0.624
14	0.25	mean	1,893,564,740	996.466	0.971	0.777	1.174	0.821	0.979	0.759	1.639	0.791	1.375	0.769	1.165	0.784
		stdev	5,731,417,083	3,011.067	1.905	0.021	2.472	0.029	1.829	0.037	3.048	0.047	3.207	0.053	2.129	0.052
		max	35,038,474,000	18,466.196	7.733	0.827	13.338	0.921	8.208	0.873	15.902	1.061	19.653	1.061	9.180	1.076
	0.50	mean	548,456,580	284.159	1.416	0.780	0.886	0.826	1.573	0.762	1.063	0.786	1.011	0.756	1.019	0.783
		stdev	1,682,656,127	854.581	3.218	0.022	1.643	0.034	3.709	0.034	2.843	0.029	1.638	0.028	1.683	0.028
		max	11,149,310,000	5,656.539	15.128	0.842	8.548	1.014	20.065	0.826	19.524	0.905	5.712	0.812	6.861	0.858
	0.75	mean	2,166,876,420	1,173.476	0.971	0.778	0.869	0.822	1.212	0.758	1.719	0.783	1.389	0.752	1.246	0.785
		stdev	5,696,359,874	3,104.532	1.167	0.021	1.635	0.031	2.436	0.025	4.074	0.020	3.216	0.036	2.302	0.023
		max	32,232,331,000	17,967.145	5.817	0.826	10.017	0.936	15.497	0.811	24.397	0.842	16.466	0.905	13.694	0.843
16	0.25	mean	23,969,479,900	13,013.183	1.479	1.018	1.551	1.062	1.129	1.006	1.567	1.018	1.904	1.006	1.628	1.001
		stdev	62,423,214,690	33,284.849	2.159	0.058	3.073	0.082	1.511	0.069	2.260	0.059	3.316	0.065	3.628	0.053
		max	415,066,585,000	218,321.058	10.408	1.186	17.458	1.419	8.698	1.217	10.348	1.155	18.556	1.139	24.566	1.154
	0.50	mean	35,306,172,400	16,400.396	3.100	0.989	2.016	0.982	1.839	0.988	1.888	0.992	1.142	0.990	1.729	0.981
		stdev	81,842,173,050	38,183.394	5.659	0.094	2.946	0.083	3.514	0.114	2.591	0.093	1.214	0.114	3.174	0.085
		max	422,599,443,000	202,490.152	26.999	1.310	11.947	1.294	22.968	1.389	10.684	1.310	4.646	1.373	18.857	1.401
	0.75	mean	29,937,133,420	13,300.289	1.592	0.993	1.099	1.011	1.746	0.987	1.371	0.988	2.052	0.994	1.525	0.990
		stdev	74,790,035,610	33,336.033	1.713	0.108	1.495	0.098	3.943	0.099	2.894	0.094	3.332	0.098	1.717	0.091
		max	437,063,466,000	193,646.647	8.942	1.295	6.419	1.248	22.977	1.264	19.577	1.326	17.412	1.248	6.491	1.279

computed the sample mean difference $\overline{D}$ and sample standard deviation s_D, as $\overline{D}=\frac{1}{50}\sum_{i=1}^{50} D_i$ and $s_D=\sqrt{\frac{\sum_{i=1}^{50}(D_i-\overline{D})^2}{49}}$. By the Central Limit Theorem (Hayter, 1996), $\overline{D}$ has a normal distribution with unknown variance and hence, the test statistic t defined as

$$t=\frac{\sqrt{50}\,\overline{D}}{s_D} \tag{7}$$

has a t-distribution with 49 degrees of freedom. Then, we can perform the t-test by constructing the hypotheses set $H_0: \mu_D = 0$; $H_1: \mu_D \neq 0$, where the hypothesis $H_1: \mu_D \neq 0$ represents the tested assumption that the performance difference of the two considered SAs is significant. Table 2 displays the t-values computed by Eq. 7 for the paired t-test. The first row is the combination of indexes representing the initial solution generation methods as defined in Section 3. Hence, each of them denotes two

Table 2. t-values for paired t-test

n	α	(11,12)	(11,21)	(11,22)	(11,31)	(11,32)	(12,21)	(12,22)	(12,31)	(12,32)	(21,22)	(21,31)	(21,32)	(22,31)	(22,32)	(31,32)
10	0.25	-0.316	-2.073	-1.730	-1.703	-1.339	-1.965	-1.678	-1.673	-1.676	0.537	1.164	1.682	0.846	1.337	0.895
	0.50	-1.439	-1.109	0.301	-0.812	-1.199	0.582	1.665	0.091	0.450	1.522	-0.301	-0.166	-0.907	-1.807	0.210
	0.75	-0.374	1.421	-0.010	2.331	1.287	1.098	0.337	1.537	1.003	-0.962	1.764	-0.229	1.531	0.905	-1.768
12	0.25	-1.158	-1.422	-0.869	-0.230	-2.136	-0.436	0.543	1.084	-1.254	0.805	1.324	-1.179	0.699	-1.358	-1.952
	0.50	-0.132	-0.609	-0.553	-0.505	1.068	-0.315	-0.431	-0.427	0.809	-0.272	0.027	1.420	0.270	0.939	1.143
	0.75	0.417	0.212	-0.532	0.466	1.152	-0.146	-0.902	0.011	0.585	-0.829	0.175	0.678	0.954	1.393	0.740
14	0.25	-0.437	-0.029	-1.293	-0.984	-0.506	0.436	-1.505	-0.341	0.028	-1.242	-0.800	-0.525	0.401	1.196	0.386
	0.50	1.008	-0.215	0.578	0.847	0.922	-1.164	-0.411	-0.368	-0.420	0.806	1.099	1.030	0.110	0.096	-0.028
	0.75	0.339	-0.629	-1.227	-0.828	-0.696	-1.528	-1.333	-1.323	-1.059	-0.740	-0.431	-0.074	0.504	0.826	0.421
16	0.25	-0.129	0.960	-0.189	-0.723	-0.245	0.831	-0.035	-0.541	-0.122	-1.172	-1.462	-0.912	-0.619	-0.096	0.386
	0.50	1.266	1.444	1.376	2.418	1.628	0.333	0.255	2.055	0.445	-0.103	1.330	0.153	1.735	0.291	-1.175
	0.75	1.468	-0.254	0.472	-0.898	0.182	-1.034	-0.627	-1.790	-1.485	0.679	-0.503	0.354	-1.709	-0.341	1.116

SAs considered for the paired t-test. For example, the value (11,12) in the third column represents two SAs using $SA\left(IS_1^A, IS_1^B\right)$ and $SA\left(IS_1^A, IS_2^B\right)$. Columns 3-17 are t-values using D_i as the difference of the ith paired values of $\%\,error$ for two SAs using the initial solution generation methods defined in the corresponding first row.

The rejection region of H_0 with a confidence level of 95% is $|t| > t_{0.95,\,49} = 2.010$. As the t-test result, we could notice that almost every t-values are less than 2.0 except 5 of the 150 cases, resulting in accepting $H_0 : \mu_D = 0$ in almost cases. More specifically, only 4 of the 12 (n, α) configurations had at most one or two t-values larger than 2.0 representing the significance of performance difference for SAs. Therefore, we could argue that the performance differences of all SAs considered are not significant.

However, we may differentiate the performance of SAs for each configuration (n, α) using the t-values as follow. If we consider two SAs using $SA\left(IS_{h1}^A, IS_{d1}^B\right)$ and $SA\left(IS_{h2}^A, IS_{d2}^B\right)$, positive t-value in Table 2 implies that the $\%\,error$ by SA using $SA\left(IS_{h1}^A, IS_{d1}^B\right)$ is larger than that by SA using $SA\left(IS_{h2}^A, IS_{d2}^B\right)$, which is equivalent to that SA using $SA\left(IS_{h2}^A, IS_{d2}^B\right)$ is better than that using $SA\left(IS_{h1}^A, IS_{d1}^B\right)$. Based on these argument, we can identify the best SA for each configuration as follows; (n, α, index for IS) = (10, 0.25, 11), (10, 0.50, 22), (10, 0.75, 31), (12, 0.25, 11), (12, 0.50, 32), (12, 0.75, 32), (14, 0.25, 11), (14, 0.50, 12), (14, 0.75, 12), (16, 0.25, 21), (16, 0.50, 31), (16, 0.75, 12), where IS is the abbreviation of initial solution.

We calculated the mean, standard deviation, and maximum values of the RDP for six SAs (i.e., $SA\left(IS_h^A, IS_d^B\right)$ $(h = 1,2,3, d = 1,2)$ as in Table 3. From the RDP values, we could verify the relative performance differences by identifying the best SA as above. Furthermore, we could argue that random generation method is better than the others for agent A because it was prominent in 6 of the 12 configurations, whereas SPT and WSPT were 2 and 4, respectively. The initial solution method for agent B did not affect the performance of $\%\,error$ because both methods (i.e., random and non-decreasing) had same number of best cases (6 in the 12 configurations).

5. Conclusions

In this paper, we considered the two-agent single-machine scheduling problem with exponential job-dependent position-based learning effects. The objective was to minimize the total weighted completion time of one agent with the restriction that the makespan of the other agent cannot exceed an upper bound. First, we suggested a B&B algorithm by developing some dominance/feasibility properties and a lower bound to find an optimal solution. Second, we designed an efficient simulated annealing (SA) algorithm to search a near optimal solution by considering six different SAs to generate initial solutions. We showed the

Table 3. Relative deviation percentages of SAs

n	α	Value of	$SA(IS_1^A, IS_1^B)$	$SA(IS_1^A, IS_2^B)$	$SA(IS_2^A, IS_1^B)$	$SA(IS_2^A, IS_2^B)$	$SA(IS_3^A, IS_1^B)$	$SA(IS_3^A, IS_2^B)$
10	0.25	mean	0.121	0.148	1.033	0.876	0.473	0.286
		stdev	0.459	0.420	3.089	3.051	1.352	0.725
		max	3.125	2.470	13.519	16.920	8.038	3.188
	0.50	mean	0.149	0.494	0.335	0.120	0.454	0.369
		stdev	0.595	1.566	0.982	0.282	2.572	1.143
		max	3.775	7.832	4.569	1.463	18.209	6.916
	0.75	mean	0.621	0.793	0.293	0.626	0.112	0.329
		stdev	1.520	3.100	0.724	2.337	0.235	0.863
		max	7.418	20.202	3.399	15.654	1.102	5.138
12	0.25	mean	0.656	1.319	1.604	1.036	0.724	2.228
		stdev	1.384	3.766	4.496	2.617	1.687	5.272
		max	8.976	21.617	23.699	13.836	10.725	23.699
	0.50	mean	0.609	0.646	0.739	0.866	0.734	0.445
		stdev	1.010	1.681	1.420	3.158	1.613	0.835
		max	3.939	10.316	5.760	21.295	9.744	3.631
	0.75	mean	0.883	0.688	0.765	1.205	0.684	0.455
		stdev	2.519	2.349	2.836	3.439	1.533	1.366
		max	15.058	15.262	18.455	20.672	8.876	6.757
14	0.25	mean	0.919	1.121	0.928	1.587	1.322	1.111
		stdev	1.885	2.461	1.799	3.059	3.199	2.096
		max	7.288	13.339	8.202	15.907	19.637	9.176
	0.50	mean	1.328	0.799	1.484	0.975	0.923	0.932
		stdev	3.203	1.641	3.707	2.837	1.539	1.669
		max	15.130	8.547	20.067	19.521	5.714	6.861
	0.75	mean	0.895	0.792	1.136	1.643	1.314	1.170
		stdev	1.168	1.602	2.422	4.053	3.231	2.289
		max	5.816	10.001	15.489	24.399	16.476	13.696
16	0.25	mean	1.349	1.421	1.000	1.437	1.773	1.498
		stdev	2.149	3.085	1.521	2.282	3.338	3.633
		max	10.407	17.341	8.674	10.269	18.564	24.394
	0.50	mean	2.935	1.851	1.675	1.723	0.978	1.565
		stdev	5.670	2.885	3.501	2.553	1.110	3.160
		max	36.999	11.147	22.984	10.684	3.905	18.861
	0.75	mean	1.407	0.914	1.556	1.178	1.859	1.338
		stdev	1.715	1.492	3.847	2.610	3.084	1.653
		max	8.936	6.366	21.38622.967	17.421	15.292	5.980

performance superiority of the suggested SA using a numerical experiment.

Using the paired t-test, we verified that there is no significant difference in the performance of $\%\, error$ between different SAs. However, we could notice that random generation method is better than the others for agent A, whereas the initial solution method for agent B did not affect the performance of $\%\, error$. Furthermore, because the CPU time by SAs, pretty faster than that by B&B, was not affected by the size n of problem instances and the tightness of upper bound U for agent B did not affect the computation time of SA procedure, we can

apply the suggested SAs in environments requiring real-time scheduling.

As the direction of future research, we can extend the current problem to the case of multi-agent scheduling problems with more than two agents. Moreover, we can consider due date in performance measure, represented as minimizing tardiness, weighted tardiness, or the number of tardy job. Because keeping due date can reflect the service level to the customer, this may be a very important issue in industry. We can also think of multi-machine scheduling problems with multi-agent, where we have to maintain information on the completion times of the jobs at each machine as the processing steps proceed, making the problem more challenging.

References

1. Agnetis, A. (2012), "Multiagent scheduling problems", *Tutorials in Operations Research, Informs 2012*, pp. 151-170.
2. Agnetis, A., Mirchandani, P.B., Pacciarelli, D., and Pacifici, A. (2004), "Scheduling problems with two competing agents", *Operations Research*, 52(2), pp. 229-242.
3. Bachman, A. and Janiak, A. (2004), "Scheduling jobs with position-dependent processing times", *Journal of the Operational Research Society*, 55, pp. 257-264.
4. Baker, K.R. and Smith, J.C. (2003), "A multiple-criterion model for machine scheduling", *Journal of Scheduling*, 6, pp. 7-16.
5. Biskup D. (1999), "Single-machine scheduling with learning considerations", *European Journal of Operational Research*, 115, pp. 173-178.
6. Biskup, D. (2008), "A state-of-the-art review on scheduling with learning considerations", *European Journal of Operational Research*, 188(2), pp. 315-329.
7. Chang, P.C., Chen, S.H., and Mani, V. (2009), "A note on due-date assignment and single machine scheduling with a learning/aging effect", *International Journal of Production Economics*, 117, pp. 142-149.
8. Cheng, T.C.E. and Wang, G. (2000), "Single machine scheduling with learning effect considerations", *Annals of Operations Research*, 98, pp. 273-290.
9. Cheng, T.C.E., Wu, W.H., Cheng, S.R., and Wu, C.C. (2011), "Two-agent scheduling with position-based deterioration jobs and learning effects", *Applied Mathematics and Computation*, 217, pp. 8804-8824.
10. Choi, J.Y. (2015), "An efficient simulated annealing for two-agent scheduling with exponential job-dependent position-based learning consideration", In the proceeding of the MISTA 2015 Conference, Prague, Czech Republic.
11. Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy, Kan AHG. (1979), "Optimization and approximation in deterministic sequencing and scheduling theory: a survey", *Annals of Discrete Mathematics*, 5, pp. 287-326.
12. Hardy, G., Littlewood, J., and Polya, G., Inequalities. London: Cambridge University Press, 1967.
13. Hayter, A. J., Probability and Statistics, International Thomson PUB, 1996.
14. Hillier, F.S. and Lieberman, G.J., Introduction to Operations Research, McGraw Hill, 2015.
15. Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). "Optimization by simulated annealing", Science, 220, pp. 671-680.
16. Kuo, W.H. and Yang, D.L. (2008), "Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect", *Journal of the Operational Research Society*, 59, pp. 416-420.
17. Lee, W.C., Wang, W.J, Shiau, Y.R., and Wu, C.C. (2010), "A single-machine scheduling problem with two-agent and deteriorating jobs", *Applied Mathematical Modelling*, 34, pp. 3098-3107.
18. Li, D.C. and Hsu, P.H. (2012), "Solving a two-agent single-machine scheduling problem considering learning effect", *Computers & Operations Research*, 39, pp. 1644-1651.
19. Liu, P., Zhou, X., and Tang, L. (2010), "Two-agent single-machine scheduling with position-dependent processing times", *International Journal of Advanced Manufacturing Technology*, 48, pp. 325-331.
20. Liu, P., Yi, N., and Zhou, X. (2011), "Two-agent single-machine scheduling problems under increasing linear deterioration", *Applied Mathematical Modelling*, 35, pp. 2290-2296.
21. Mosheiov, G. (2001), "Parallel machine scheduling with a learning effect", *Journal of the Operational Research Society*, 52, pp. 1165-1169.
22. Mosheiov, G. (2005), "A note on scheduling deteriorating jobs", *Mathematical and Computer Modelling*, 41, pp. 883-886.
23. Moshiov, G. and Sidney, J. B. (2003), "Scheduling with general job-dependent learning curves", *European Journal of Operational Research*, 147, pp. 665-670.
24. Wang, J.B. and Xia, Z.Q. (2005), "Flow-shop scheduling with a learning effect", *Journal of the Operational*

Research Society, 56, pp. 1325-1330.
25. Wu, C.C., Huang, S.K., and Lee, W.C. (2011), "Two-agent scheduling with learning consideration", *Computers & Industrial Engineering*, 61, pp. 1324-1335.
26. Wu, W.H., Xu, J., Wu, W.H., Yin, Y., Cheng, I.F., and Wu, C.C. (2013), "A tabu method for a two-agent single-machine scheduling with deterioration jobs", *Computers & Operations Research*, 40, pp. 2116-2127.

최 진 영 (choijy@ajou.ac.kr)

1991 한양대학교 산업공학과 학사
1993 KAIST 산업공학과 석사
2004 Georgia Tech 산업및시스템공학과 공학박사
1993～1999년 한국전자통신연구소 선임연구원
2005～2007년 삼성네트웍스 부장
2007～현재 아주대학교 부교수

관심분야 : 스케쥴링, 데이터 Analytics, 프로세스 마이닝, 모델링 & 시뮬레이션