Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 806325, 13 pages
http://dx.doi.org/10.1155/2013/806325

Research Article

Two-Agent Single-Machine Scheduling of Jobs with Time-Dependent Processing Times and Ready Times

Jan-Yee Kung,[1] Yuan-Po Chao,[1] Kuei-I Lee,[2] Chao-Chung Kang,[3] and Win-Chin Lin[4]

[1] *Department of Business Administration, Cheng Shiu University, Kaohsiung 1037, Taiwan*
[2] *Department of Hospitality Management, Tunghai University, Taichung 1001, Taiwan*
[3] *Department of Business Administration and Graduate Institute of Management, Providence University, Taichung 1008, Taiwan*
[4] *Department of Statistics, Feng Chia University, Taichung 1007, Taiwan*

Correspondence should be addressed to Yuan-Po Chao; rainbow@csu.edu.tw

Received 22 May 2013; Accepted 18 June 2013

Academic Editor: Yunqiang Yin

Scheduling involving jobs with time-dependent processing times has recently attracted much research attention. However, multiagent scheduling with simultaneous considerations of jobs with time-dependent processing times and ready times is relatively unexplored. Inspired by this observation, we study a two-agent single-machine scheduling problem in which the jobs have both time-dependent processing times and ready times. We consider the model in which the actual processing time of a job of the first agent is a decreasing function of its scheduled position while the actual processing time of a job of the second agent is an increasing function of its scheduled position. In addition, each job has a different ready time. The objective is to minimize the total completion time of the jobs of the first agent with the restriction that no tardy job is allowed for the second agent. We propose a branch-and-bound and several genetic algorithms to obtain optimal and near-optimal solutions for the problem, respectively. We also conduct extensive computational results to test the proposed algorithms and examine the impacts of different problem parameters on their performance.

1. Introduction

In many scheduling models researchers assume that the job processing times are known and fixed parameters. However, the job processing times can be prolonged due to deterioration or shortened due to learning over time in real-life situations. Browne and Yechiali [1] present fire-fighting as an example of job deterioration while Biskup [2] cites workers' skill improvement as an example of job learning. Both examples and their corresponding situations are referred to as "time-dependent scheduling" in the literature.

Time-dependent scheduling was first introduced by J. N. D. Gupta and S. K. Gupta [3] and Browne and Yechiali [1] for deterioration jobs, whereas by Biskup [2] for the learning effect. Since then, a lot of scheduling models involving job time-dependent processing times have been proposed from a variety of perspectives. For more detailed reviews of scheduling problems with deteriorating jobs, we refer the reader to Alidaee and Womer [4] and Cheng et al. [5], and for a detailed review of scheduling problems with learning effects, we refer the reader to Biskup [6]. More recently, scheduling research that considers both deteriorating jobs and learning effects has become popular. For details on this stream of scheduling research, the reader may refer to Wang [7], Wang and Cheng [8, 9], Wang and Liu [10], Wang and Guo [11], Zhang and Yan [12], Yang [13], and Zhu et al. [14], among others.

In the past, the scheduling literature was dominated by studies that deal with a single criterion. In reality, jobs might come from several agents (customers) that have different requirements to meet. However, scheduling research in the multiple-agent setting involving jobs with time-dependent processing times is relatively unexplored. Among the few studies on this topic, Liu and Tang [15] studied multiagent scheduling with deteriorating jobs in which they assumed that the actual processing time of job J_j is $\delta_j t$, where $\delta_j > 0$ and t denote the deteriorating rate and starting time of J_j, respectively. Liu et al. [16] considered two two-agent problems with position-dependent processing times. In the aging-effect model, they assumed that the actual processing time of J_j is $p_j + br$ ($p_j + bv$) if it belongs to agent $A(B)$.

Meanwhile, in the learning-effect model, they assumed that the actual processing time of J_j is $p_j - br$ ($p_j - bv$) if it belongs to agent $A(B)$, where r (or v) denotes the job position and b, $b > 0$, is the aging or learning effect. Cheng et al. [17] considered a two-agent scheduling problem in which they assumed that, given a schedule, the actual processing time of a job of the first agent is a function of position-based learning while the actual processing time of a job of the second agent is a function of position-based deterioration. The objective is to minimize the total weighted completion time of the jobs of the first agent with the restriction that no tardy job is allowed for the second agent. Wu et al. [18] studied two-agent scheduling in which the actual processing time of J_j is $p_j(1 + \sum_{l=1}^{r-1} p_{[l]})^a$ ($p_j(1 + \sum_{l=1}^{r-1} p_{[l]})^\beta$) if it is a job of AG_0 (AG_1) scheduled in the rth position of a sequence. Their objective function is to find an optimal schedule to minimize $\sum_{j=1}^{n} w_j C_j(S)(1 - I_j)$, subject to $\max_{1 \le j \le n}\{L_j(S)I_j\} \le 0$. They proposed branch-and-bound and ant colony algorithms to solve the problem. However, they ignored job ready times. Wu et al. [19] considered a single-machine problem with the sum-of-processing time based learning effect and release times, where the objective is to minimize the total weighted completion time. They assumed that the actual processing time of job j is $p_{jr} = p_j(1 - \sum_{l=1}^{r-1} p_{[l]} / \sum_{l=1}^{n} p_l)^a$ if it is scheduled in the rth position, where $a \ge 1$ is a learning ratio common to all the jobs. They proposed a branch-and-bound algorithm and a genetic heuristic-based algorithm to treat the problem. Wu et al. [18] used a branch-and-bound and a ant colony algorithm to solve a two-agent scheduling with learning and deteriorating jobs. Lun et al. [20] and Zhang et al. [21] gave applications of multiagent scheduling of jobs with ready times in the shipping industry. Specifically, ships belonging to shipping companies (multiple agents) call at a port at different times. The port needs to find a suitable schedule to serve the ships. In this context, the port is the single machine and the arriving ships from different shipping companies are jobs belonging to different agents with ready times. Inspired by this and other applications, we study in this paper a two-agent single-machine scheduling problem in which the jobs have both time-dependent processing times and ready times. We consider the model in which the actual processing time of a job of the first agent is a decreasing function of its scheduled position while the actual processing time of a job of the second agent is an increasing function of its scheduled position. The objective is to minimize the total completion time of the jobs of the first agent with the restriction that no tardy job is allowed for the second agent.

The remainder of this paper is organized as follows. We present the problem formulation in the next section. In Section 3 we discuss the computational complexity of the problem. In Section 4 we develop some dominance properties and a lower bound to enhance the search efficiency for the optimal solution, followed by introducing a branch-and-bound and several genetic algorithms. In Section 5 we present the results of computational experiments conducted to assess the performance of the proposed algorithms. We conclude the paper and suggest some topics for future research in the last section.

2. Model Formulation

We formulate the scheduling problem under study as follows. There are n jobs for processing on a single machine. Each job J_j becomes available for processing at time $r_j \ge 0$. Each job belongs to one of two agents, namely, AG_0 and AG_1. For job J_j, there is a normal processing time p_j and an agent code I_j, where $I_j = 0$ if $J_j \in AG_0$ and $I_j = 1$ if $J_j \in AG_1$. We assume that all the jobs of AG_0 have a position-based learning rate a with $a \le 0$, while all the jobs of AG_1 have a position-based deteriorating rate b with $b \ge 0$. Under the proposed model, the actual processing time of J_j is $p_j r^a$ ($p_j r^b$) if it belongs to AG_0 (AG_1) and is scheduled in position r of a sequence.

For a given schedule S, let $C_j(S)$ be the completion time of J_j, and $U_j(S) = 1$ if $C_j(S) > d_j$ and zero otherwise. The objective is to find an optimal schedule to minimize $\sum_{j=1}^{n} C_j(S)(1 - I_j)$, subject to $\sum_{j=1}^{n} U_j(S)I_j = 0$. Adopting the three-field notation scheme $\alpha|\beta|\gamma^A : \gamma^B$ introduced by Agnetis et al. [22], we denote the problem as $1|r_j, p_{jr} = p_j r^x, x \in \{a, b\}| \sum_{j=1}^{n} C_j(S)(1 - I_j) : \sum_{j=1}^{n} U_j(S)I_j = 0$.

As for the major results of research on multiagent scheduling without learning effects or deteriorating jobs, the reader may refer to Baker and Smith [23], Agnetis et al. [22], Yuan et al. [24], Cheng et al. [25, 26], Ng et al. [27], Agnetis et al. [28], Mor and Mosheiov [29, 30], Yin et al. [31, 32], and so forth.

3. Branch-and-Bound Algorithm

Due to the fact that our proposed problem is an NP-hard one (see [33]), in this section we apply the branch-and-bound technique to search for the optimal solution. In order to facilitate the searching process, we first develop some dominance properties, followed by presenting a lower bound to fathom the searching tree. We then present the procedures of branch-and-bound algorithm and suggest some heuristics.

3.1. Dominance Rules. Assume that schedule S has two adjacent jobs J_i and J_j with J_i immediately preceding J_j. Perform a pairwise interchange of J_i and J_j to derive a new sequence S'. In addition, assume that J_i and J_j are in the rth and $(r + 1)$th positions of S and that the starting time to process job J_i in S is t.

Lemma 1. *If $J_i, J_j \in AG_0$, $r_j \ge \max\{r_i, t\} + p_i r^a$, then S dominates S'.*

Proof. The completion times of jobs J_i and J_j in S and S' are, respectively,

$$\begin{aligned} C_i(S) &= \max\{r_i, t\} + p_i r^a, \\ C_j(S) &= \max\{\max\{r_i, t\} + p_i r^a, r_j\} + p_j (r+1)^a, \\ C_j(S') &= \max\{r_j, t\} + p_j r^a, \\ C_i(S') &= \max\{\max\{r_j, t\} + p_j r^a, r_i\} + p_i (r+1)^a, \end{aligned} \tag{1}$$

From $r_j \geq \max\{r_i, t\} + p_i r^a$, we have

$$C_i\left(S'\right) > C_j\left(S'\right) = \max\left\{r_j, t\right\} + p_j r^a > r_j + p_j (r+1)^a = C_j(S) > C_i(S). \quad (2)$$

It follows that $C_i(S') + C_j(S') > C_j(S) + C_i(S)$, so S dominates S'. □

Lemma 2. *If $J_i, J_j \in AG_0$, $\max\{r_i, t\} \leq r_j < \max\{r_i, t\} + p_i r^a$ and $p_i < p_j$, then S dominates S'.*

Proof. By the assumption and Lemma 1, the completion times of jobs J_i and J_j in S and S' can be reformulated as

$$\begin{aligned} C_i(S) &= \max\left\{r_i, t\right\} + p_i r^a, \\ C_j(S) &= \max\left\{r_i, t\right\} + p_i r^a + p_j (r+1)^a, \\ C_j\left(S'\right) &= \max\left\{r_j, t\right\} + p_j r^a, \\ C_i\left(S'\right) &= \max\left\{r_j, t\right\} + p_j r^a + p_i (r+1)^a, \end{aligned} \quad (3)$$

respectively. From $\max\{r_i, t\} \leq r_j$ and $p_i < p_j$, it is easy to see that $C_i(S') > C_j(S)$ and $C_j(S') > C_i(S)$. It follows that S dominates S'. □

Lemma 3. *If $J_i, J_j \in AG_0$, $\max\{r_j, t\} \leq r_i < \max\{r_j, t\} + p_j r^a$, and $(p_j - p_i)(r^a - (r+1)^a) \geq r_i - \max\{r_j, t\}$, then S dominates S'.*

Proof. The proof is similar to that of Lemma 2. □

Lemma 4. *If $J_i, J_j \in AG_0$, $\max\{r_i, r_j\} \leq t$, and $p_i < p_j$, then S dominates S'.*

Proof. The proof is similar to that of Lemma 2. □

Lemma 5. *If $J_i, J_j \in AG_1$, $\max\{r_i, t\} + p_i r^b \leq d_i < \max\{\max\{r_j, t\} + p_j r^b, r_i\} + p_i (r+1)^b$, and $\max\{\max\{r_i, t\} + p_i r^b, r_j\} + p_j (r+1)^b \leq d_j$, then S dominates S'.*

Proof. The completion times of jobs J_i and J_j in S and S' are, respectively,

$$\begin{aligned} C_i(S) &= \max\left\{r_i, t\right\} + p_i r^b, \\ C_j(S) &= \max\left\{\max\left\{r_i, t\right\} + p_i r^b, r_j\right\} + p_j (r+1)^b, \\ C_j\left(S'\right) &= \max\left\{r_j, t\right\} + p_j r^b, \\ C_i\left(S'\right) &= \max\left\{\max\left\{r_j, t\right\} + p_j r^b, r_i\right\} + p_i (r+1)^b. \end{aligned} \quad (4)$$

The given conditions lead to $C_i(S) \leq d_i < C_i(S')$ and $C_j(S) \leq d_j$, implying that schedule S' is infeasible. Hence, S dominates S'. □

Lemma 6. *If $J_i, J_j \in AG_1$ and $\max\{\max\{r_j, t\} + p_j r^b, r_i\} + p_i (r+1)^b < \max\{\max\{r_i, t\} + p_i r^b, r_j\} + p_j (r+1)^b \leq \min\{d_i, d_j\}$, then S dominates S'.*

Proof. The given condition implies that $U_i(S') + U_j(S') = U_i(S) + U_j(S) = 0$ and $C_i(S') > C_j(S)$, so S dominates S'. □

The proofs of Lemmas 7 to 9 are omitted since they are similar to those of Lemmas 1 and 2.

Lemma 7. *If $J_i \in AG_0$, $J_j \in AG_1$, $r_i < \max\{r_j, t\} + p_j r^b$, and $\max\{\max\{r_i, t\} + p_i r^a, r_j\} + p_j (r+1)^b \leq \min\{\max\{r_j, t\} + p_j r^b + p_i (r+1)^a, d_j\}$, then S dominates S'.*

Lemma 8. *If $J_i \in AG_1$, $J_j \in AG_0$ and $\max\{r_i, t\} + p_i r^b \leq d_i < \max\{\max\{r_j, t\} + p_j r^a, r_i\} + p_i (r+1)^b$, then S dominates S'.*

Lemma 9. *If $J_i \in AG_1$, $J_j \in AG_0$ and $\max\{r_i, t\} + p_i r^b \leq \min\{d_i, r_j\}$, then S dominates S'.*

Next, we present two lemmas to determine the feasibility of a partial sequence. Let (π, π') be a sequence of jobs where π is the scheduled part with k jobs and π' is the unscheduled part. Moreover, let $C_{[k]}$ be the completion time of the last job in π.

Lemma 10. *If there is a job $J_j \in AG_1 \cap \pi'$ such that $\max\{C_{[k]}, r_j\} + p_j (k+1)^b > d_j$, then sequence (π, π') is not a feasible solution.*

Lemma 11. *If all the unscheduled jobs belong to AG_0 and there exists a job $J_j \in \pi'$ such that $\max\{C_{[k]}, r_j\} + p_j (k+1)^a \leq r_k$ for all job $J_k \in \pi' \setminus \{J_j\}$, then job J_j may be assigned to the $(k+1)$th position.*

Lemma 12. *If all the unscheduled jobs belong to AG_0 and $\max\{r_j\}_{J_j \in \pi'} \leq t$, then the shortest processing time (SPT) rule gives an optimal sequence for the remaining unscheduled jobs.*

3.2. A Lower Bound for $1|r_j, p_{jr}^A = p_j^A r^a, p_{jr}^B = p_j^B r^b| \sum C_j^A : \sum U_j^B \leq 0$. The efficiency of the branch-and-bound algorithm depends greatly on the lower bounds for the partial sequences. In this subsection we propose a lower bound. Let PS be a partial schedule in which the order of the first k jobs is determined and US be the unscheduled part with $(n-k)$ jobs, where there are n_0 jobs belonging to AG_0 and n_1 jobs belonging to AG_1 with $n_0 + n_1 = n - k$. Moreover, let $p_{[j]}, r_{[j]}$, and $C_{[j]}$ denote the normal processing time, release time, and completion time of the jth job in a sequence, respectively, where $j = \{1, 2, \ldots, n\}$. A lower bound for the partial sequence PS is obtained by scheduling the jobs belonging to AG_0 first in the SPT order and then scheduling the jobs belonging to AG_1 in any order. Then the completion time of the $(k+1)$th job is

$$\begin{aligned} C_{[k+1]} &= \max\left\{r_{[k+1]}, C_{[k]}\right\} + p_{[k+1]} (k+1)^a \\ &\geq C_{[k]} + p_{[k+1]} n^a. \end{aligned} \quad (5)$$

Similarly, the completion time for the $(k+2)$th job is

$$C_{[k+1]} = \max\{r_{[k+2]}, C_{[k+1]}\} + p_{[k+2]}(k+2)^a \geq C_{[k+1]} + p_{[k+2]}n^a. \quad (6)$$

Continuing in this fashion, the completion time of the $(k+l)$th job is

$$C_{[k+l]} = \max\{r_{[k+l]}, C_{[k+l-1]}\} + p_{[k+l]}(k+l)^a \geq C_{[k+l-1]} + p_{[k+l]}n^a. \quad (7)$$

Based on the above analysis, a lower bound for partial sequence PS can be calculated as follows.

Algorithm 13. Step 1. Sort the jobs of AG_0 in nondecreasing order of their processing times, that is, $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(n_0)}$.

Step 2. Calculate $\overline{C}_{[k+l]} = C_{[k]} + n^a \sum_{j=1}^{l} p_{[k+j]}$ for $1 \leq l \leq n_1$.

Therefore, a lower bound for the partial sequence PS is

$$\mathrm{LB}_1 = \sum_{j \in \mathrm{PS}} C_{[j]}\left(1 - I_j\right) + \sum_{l=1}^{n_1} \overline{C}_{[k+l]}. \quad (8)$$

However, this lower bound may not be tight if the release times are large. To overcome this situation, we propose a second lower bound by taking account of the ready times. By definition, the completion time of the $(k+1)$th job is

$$C_{[k+1]} = \max\{r_{[k+1]}, C_{[k]}\} + p_{[k+1]}(k+1)^a \geq r_{[k+1]} + p_{[k+1]}n^a. \quad (9)$$

Similarly, the completion time for the $(k+2)$th job is

$$C_{[k+1]} = \max\{r_{[k+2]}, C_{[k+1]}\} + p_{[k+2]}(k+2)^a \geq r_{[k+2]} + p_{[k+2]}n^a. \quad (10)$$

Continuing in this fashion, the completion time of the $(k+l)$th job is

$$C_{[k+l]} = \max\{r_{[k+l]}, C_{[k+l-1]}\} + p_{[k+l]}(k+l)^a \geq r_{[k+l]} + p_{[k+l]}n^a. \quad (11)$$

Based on the above analysis, another lower bound for the partial sequence PS can be calculated as follows:

$$\mathrm{LB}_2 = \sum_{j \in \mathrm{PS}} C_{[j]}\left(1 - I_j\right) + \sum_{l=1}^{n_1} \left(r^A_{[k+l]} + p^A_{[k+l]}n^a\right). \quad (12)$$

In order to make the lower bound tighter, we choose the maximum value between (8) and (12) as the lower bound for PS. That is,

$$\mathrm{LB} = \max\{\mathrm{LB}_1, \mathrm{LB}_2\}. \quad (13)$$

3.3. Genetic Algorithms. Genetic algorithm (GA) is a metaheuristic method that is commonly used to tackle combinatorial optimization problems [34]. A genetic algorithm starts with a set of feasible solutions (population) and iteratively replaces the current population by a new population. It requires a suitable encoding for the problem and a fitness function measures the quality of each encoded solution (chromosome or individual). The reproduction mechanism selects the parents and recombines them using a crossover operator to generate offspring that are submitted to a mutation operator in order to alter them locally [35]. The main steps of the GA are summarized in the following.

3.3.1. Representation of Structure. In this paper we adopt a structure as a sequence of the jobs of the problem based on the method by Etiler et al. [36].

3.3.2. Initial Population. To get the final solution more quickly, we construct the initial population by using three heuristics [37]. We propose the use of three initial sequences. We generate the first initial sequence by arranging the jobs of AG_1 in the earliest due date (EDD) order, followed by arranging the jobs of AG_0 in the smallest SPT order (recorded as GA_1), followed by arranging the jobs of AG_0 in the earliest ready times (ERT) first order (recorded as GA_2), and followed by arranging the jobs of AG_0 in the EDD order (recorded as GA_3).

3.3.3. Population Size. Following Chen et al. [38], we use an initial population as one schedule and create other members by applying interchange mutation until the number of members is equal to the population size. We set the population size (say, N) equal to the number of jobs (i.e., $N = n$) based on preliminary tests.

3.3.4. Fitness Function. Given that the objective of the problem is to minimize the total completion time, we define the fitness function of the strings as follows:

$$f\left(S_i(v)\right) = \max_{1 \leq l \leq N}\left\{\sum_{j=1}^{n} C_j\left(S_l(v)\right)\right\} - \sum_{j=1}^{n} C_j\left(S_i(v)\right), \quad (14)$$

where $S_i(v)$ is the ith string chromosome in the vth generation, $\sum_{j=1}^{n} C_j(S_i(v))$ is the total completion time of $S_i(v)$, and $f(S_i(v))$ is the fitness function of $S_i(v)$. Therefore, the probability of selection of a schedule $P(S_i(v))$ is to ensure that the probability of selection of a sequence with a lower value of the objective function is higher, which is defined as follows:

$$P\left(S_i(v)\right) = \frac{f\left(S_i(v)\right)}{\sum_{l=1}^{N} f\left(S_l(v)\right)}. \quad (15)$$

3.3.5. Crossover. Crossover is used to generate a new offspring from two parents. We adopt the partially matched crossover method, which is commonly used in GA [36]. In order to preserve the best schedule that has the minimum total completion time in each generation, we keep it to the

next population with no change. This operation enables us to choose a higher crossover with the crossover rate $P_c = 1$.

3.3.6. Mutation. Mutation is used to prevent premature falling into a local optimal in the GA procedure. It can be considered as a transition from a current solution to its neighbourhood solution in a local search algorithm. In this study we set the mutation rate P_m at 1.0 based on preliminary experiments.

3.3.7. Selection. In this paper we fix the population sizes at n from generation to generation. Excluding the best schedule that has the minimum total completion time, the rest of the offspring are generated from the parent chromosomes by the roulette wheel method.

3.3.8. Stopping Rule. We end the procedure of the GA after $10 * n$ generations based on preliminary experiments.

4. Computational Results

We carried out computational experiments to assess the performance of proposed branch-and-bound and genetic algorithms over a range of problem parameters. We coded all the algorithms in FORTRAN using Compaq Visual Fortran version 6.6 and conducted the experiments on a personal computer powered by an Intel Pentium(R) Dual-Core CPU E6300 @ 2.80 GHz with 2 GB RAM operating under Windows XP. We generated the job processing times from a uniform distribution $U(1, 100)$. Following the design of Reeves [37], we generated the ready times of the jobs from another uniform distribution $U(0, 20n\lambda)$, where n is the number of jobs and λ is a control parameter. In our tests we set the value of λ at $1/n$, 0.25, 0.5, 0.75, and 1. In addition, following the design of Fisher [39], we generated the due dates of the jobs of AG_1 from a uniform distribution $T \times U(1 - \tau - R/2, 1 + \tau + R/2)$, where T is the sum of the normal processing times of the n jobs; that is, $T = \sum_{i=1}^{n} p_i$, τ took the values 0.25 and 0.5, while R took the values 0.25, 0.5, and 0.75. We fixed the proportion of the jobs of agent AG_1 at pro = 0.5 in the experiments.

For the branch-and-bound algorithm, we recorded the average and maximum numbers of nodes, as well as the average (mean) and maximum of the execution times (in seconds). For the proposed GA algorithms, we recorded the mean and maximum percentage errors. We calculate the percentage error of a solution produced by a heuristic algorithm as

$$\frac{V - V^*}{V^*} \times 100\%, \tag{16}$$

where V and V^* are the total completion time of the heuristic and the optimal solution of the jobs of the first agent, respectively. We did not record the computational times of the GA algorithms because they all were less one second to obtain a solution.

We carried out the computational experiments in two parts. For the first part of the experiments, we tested instances

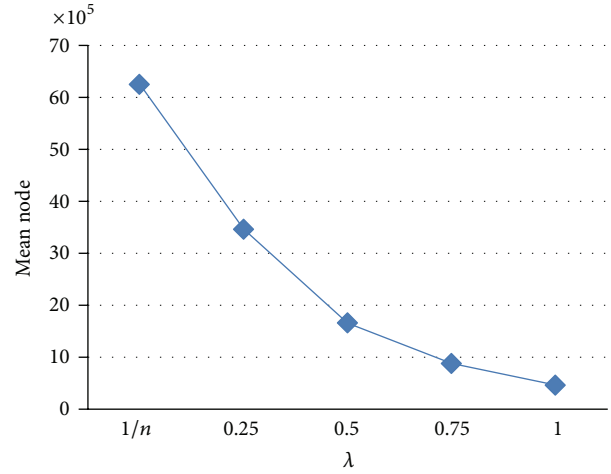

Figure 1: The behavior of λ at $n = 14$.

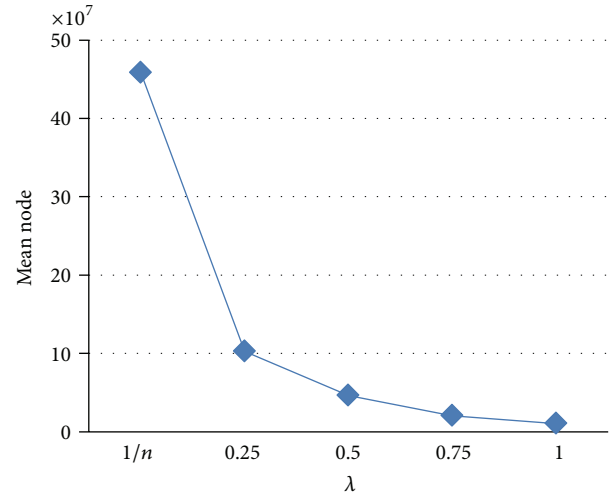

Figure 2: The behavior of λ at $n = 16$.

of the problem at $n = 14$ and 16. Moreover, we took three different values of the learning effect 70%, 80%, and 90% (corresponding to $\alpha = -0.515, -0.322$, and -0.152, resp.) and three different values of the deteriorating effect 70%, 80%, and 90% (corresponding to $\beta = 0.515$, 0.322, and 0.152, resp.). We randomly tested a set of 100 instances for each case. As a result, we examined 54,000 instances. The instances with numbers of nodes few than 10^8 were recorded as instance solved or IS. We further extracted the relative results to report in the following.

As regards the performance of the branch-and-bound algorithm, we see from Figures 1, 2, 3, 4, 5, and 6 that the number of nodes declines as the value of λ, τ, or R increases no matter whether $n = 14$ or 16. The reason is due to the fact that our proposed dominance rules and lower bound are more effective at a bigger value of λ, τ, or R.

We observe from Figures 7 and 8 the performance trend of the learning effect. As shown in Figures 9 and 10, the instances with larger deteriorating values are easier to solve than those with smaller deteriorating values.

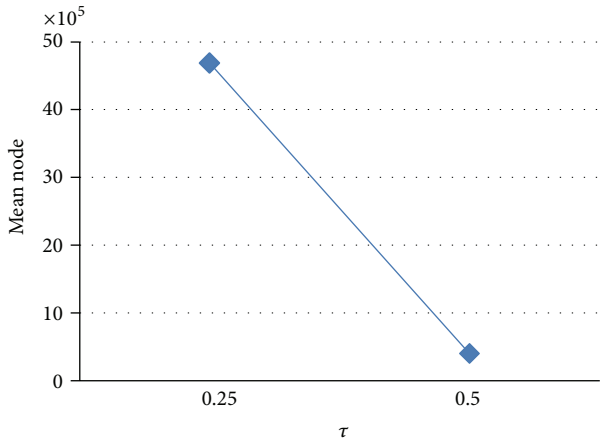

Figure 3: The behavior of τ at $n = 14$.

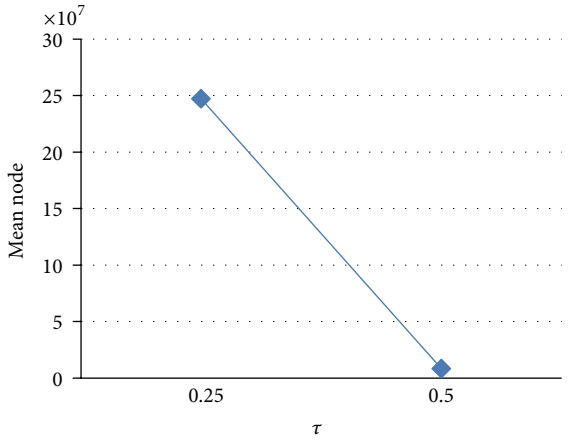

Figure 4: The behavior of τ at $n = 16$.

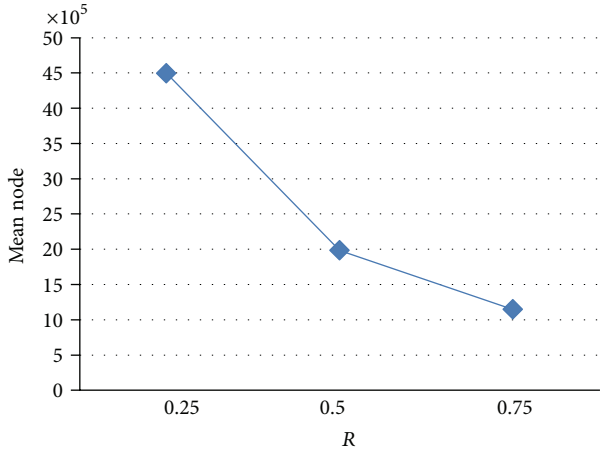

Figure 5: The behavior of R at $n = 14$.

We also see that the branch-and-bound algorithm generates more nodes for instances with a bigger value of λ, τ, or R. This also implies that the number of IS at a bigger value of λ, τ, or R is higher than that at a smaller value (i.e., see Figures 11 and 12).

Moreover, Figures 13 and 14 show that the instances with a weaker learning effect are easier to solve than those with

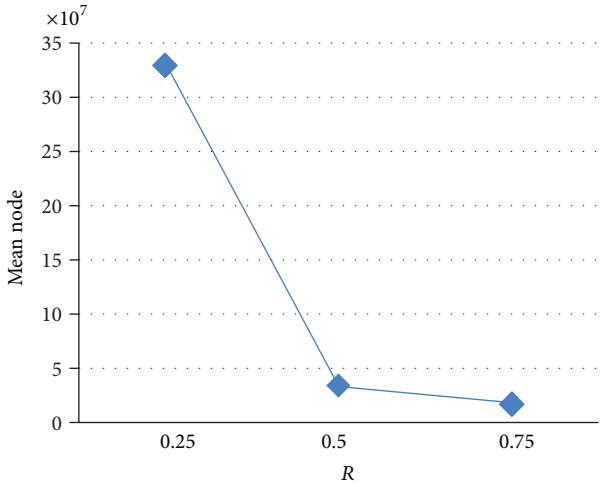

Figure 6: The behavior of R at $n = 16$.

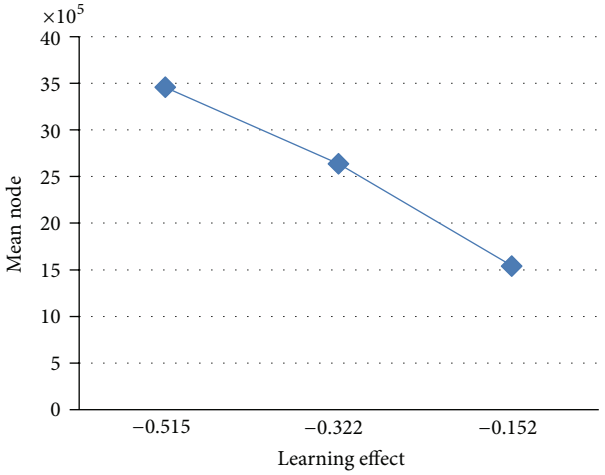

Figure 7: The behavior of a at $n = 14$.

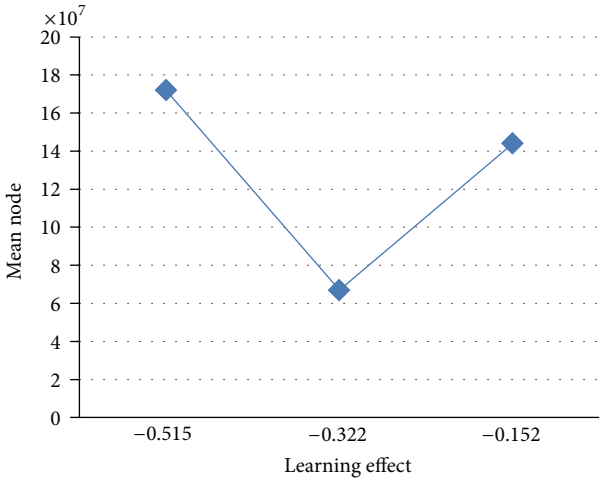

Figure 8: The behavior of a at $n = 16$.

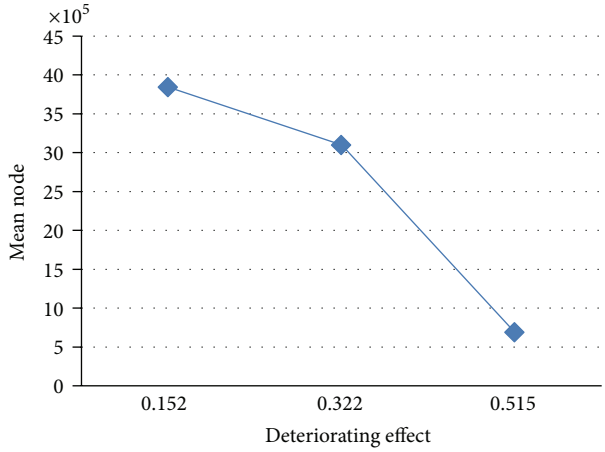

Figure 9: The behavior of β at $n = 14$.

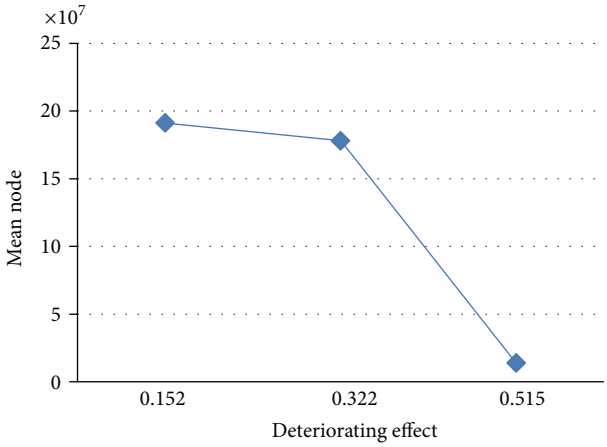

Figure 10: The behavior of β at $n = 16$.

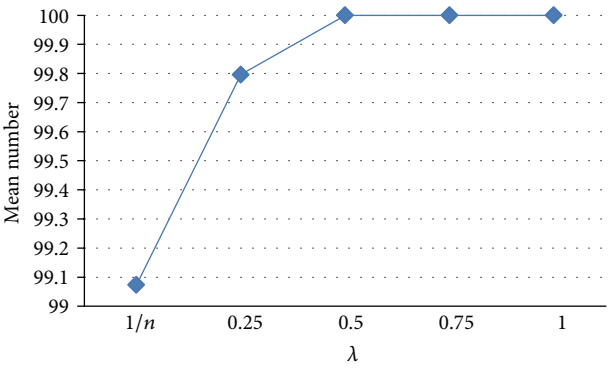

Figure 11: The behavior of IS as λ changes at $n = 14$.

a stronger learning effect, whereas Figures 15 and 16 show that the deteriorating effect keeps the same trend of IS. The performance of the branch-and-bound algorithm in terms of CPU time over the range of problem parameters tested is similar to that in terms of number of nodes generated.

For the performance of the GA heuristics, Figures 17, 18, and 19 show that when $n = 14$, the mean percentage errors of

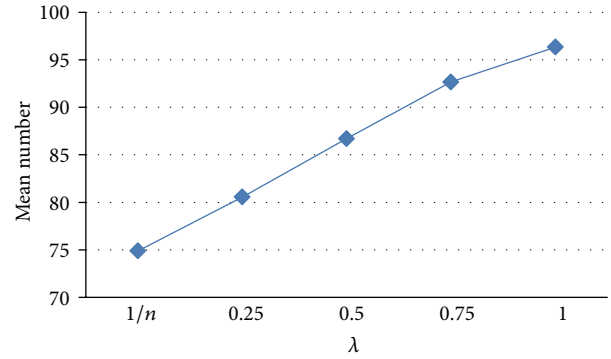

Figure 12: The behavior of IS as λ changes at $n = 16$.

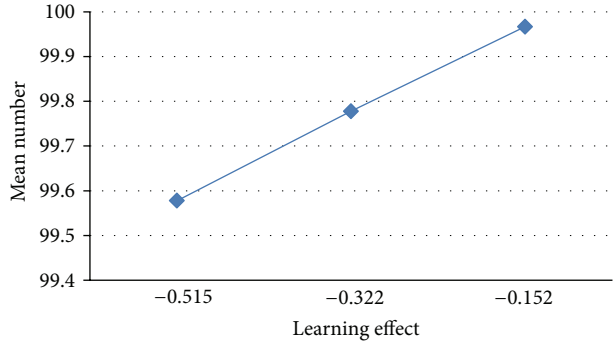

Figure 13: The behavior of IS as a changes at $n = 14$.

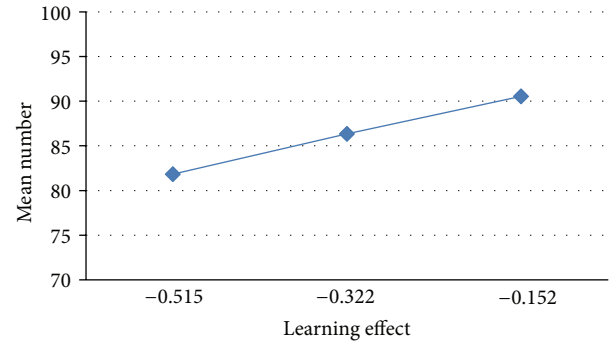

Figure 14: The behavior of IS as a changes at $n = 16$.

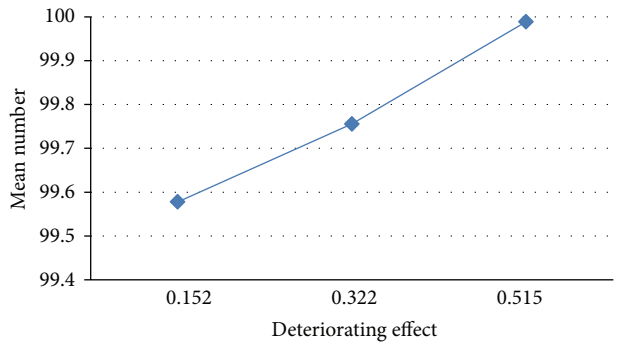

Figure 15: The behavior of IS as β changes at $n = 14$.

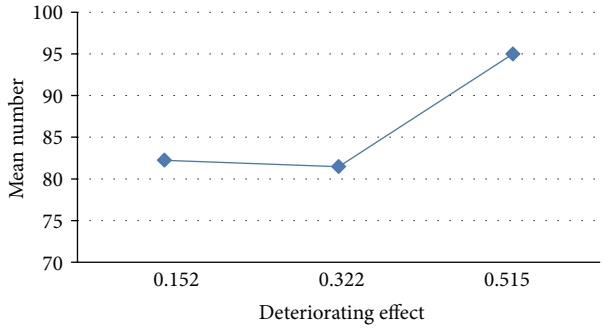

Figure 16: The behavior of IS as β changes at $n = 16$.

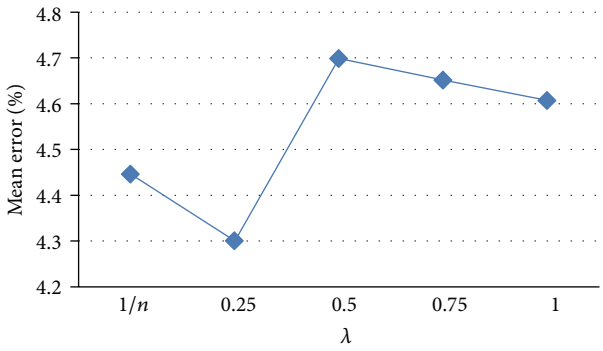

Figure 17: The behavior of GA_1 as λ varies.

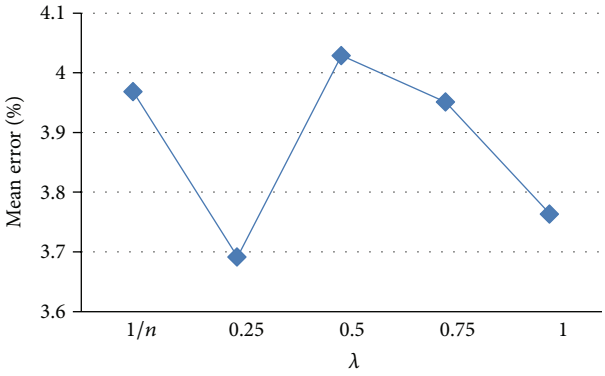

Figure 18: The behavior of GA_2 as λ varies.

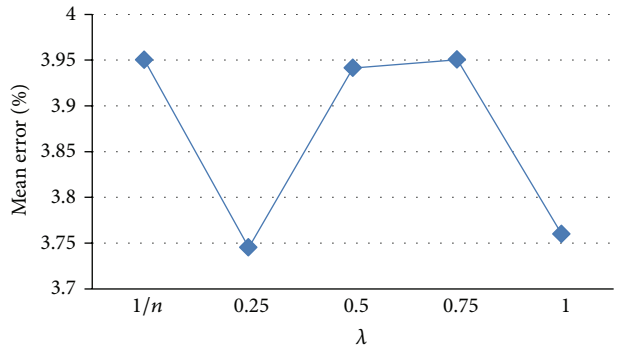

Figure 19: The behavior of GA_3 as λ varies.

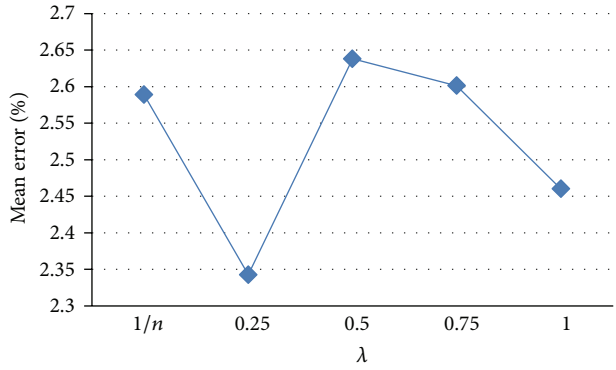

Figure 20: The behavior of GA^* as λ varies.

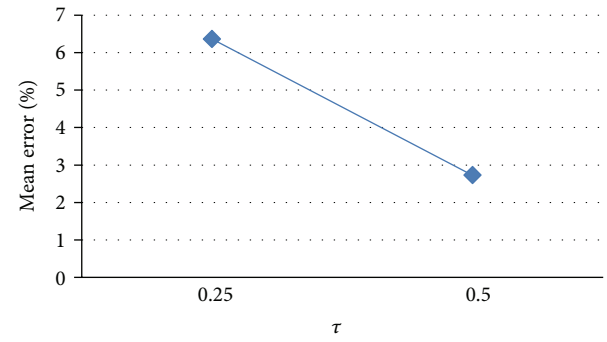

Figure 21: The behavior of GA_1 as τ varies.

GA_1, GA_2, and GA_3 are within the ranges 4.3%–4.7%, 3.7%–4.1%, and 3.75%–3.95%, respectively, regardless of value of λ. Since all the GAs only take less than a second of CPU time to obtain a solution, we further take $\min\{GA_i, i = 1, 2, 3\}$ as GA^*.

We observe from Figure 20 that the mean percentage error of GA^* declines to within the range 2.3%–2.7%. When $n = 14$, Figures 21, 22, 23, and 24 show that at a bigger value of τ ($\tau = 0.5$), GA_1, GA_2, GA_3, and GA^* yield smaller percentage errors than at a smaller value of τ ($\tau = 0.25$); however, Figures 25, 26, 27, and 28 show that at a smaller value of R, GA_1, GA_2, GA_3, and GA^* yield smaller percentage errors than at a bigger value of R.

As regards the impacts of learning or deteriorating on the proposed GAs, Figures 29, 30, 31, and 32 show that all the GAs perform better at a weaker learning effect than a stronger one, but the performance reverses under the deteriorating effect as shown in Figures 33, 34, 35, and 36.

Overall, we observe from Figures 37 and 38 that the grand means of the mean error percentages of GA_2 and GA_3 are smaller than those of GA_1 when $n = 12$ and 16, respectively.

The result also shows that GA^* performs well and keeps about 3% of the grand means of the mean error percentages, which is clearly lower than those of GA_1, GA_2, and GA_3.

In the second part of the experiments, we further assessed the performance of the proposed GA algorithms in solving instances with large numbers of jobs. We set n at 30 and 40 and fixed the parameters as follows: τ took the values of 0.25 and 0.5, while R took the values of 0.25, 0.50, and 0.75. We fixed the proportion of the jobs of agent AG_1 at

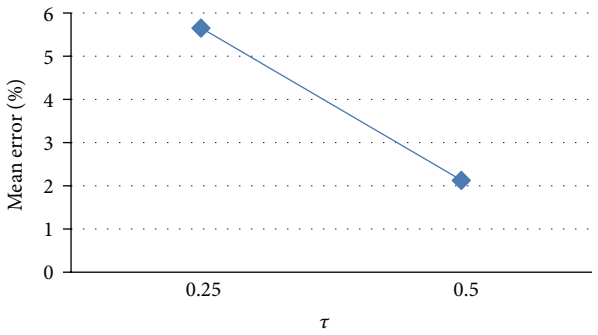

Figure 22: The behavior of GA_2 as τ varies.

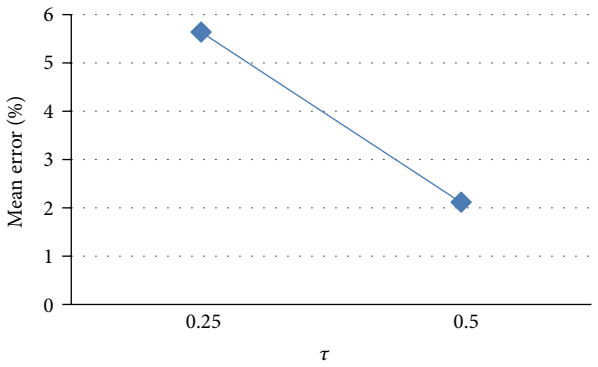

Figure 23: The behavior of GA_3 as τ varies.

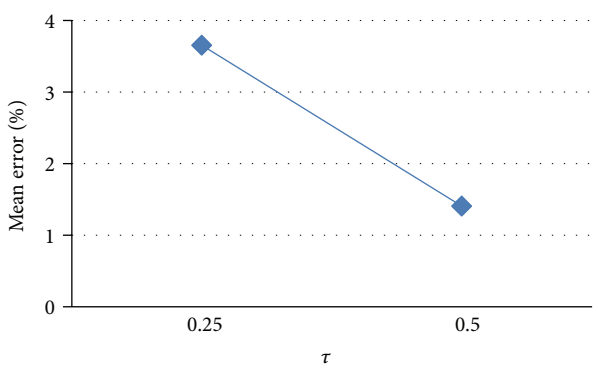

Figure 24: The behavior of GA^* as τ varies.

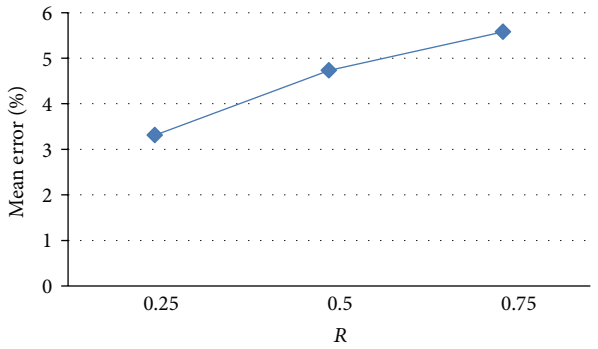

Figure 25: The behavior of GA_1 as R varies.

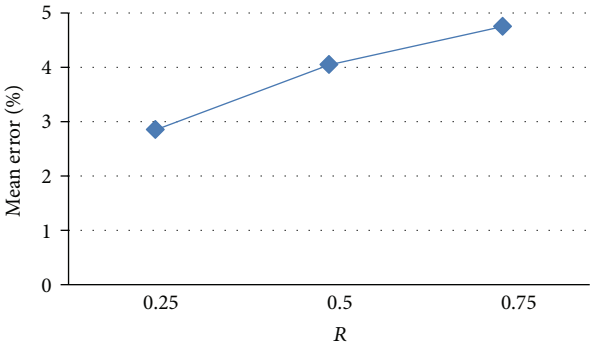

Figure 26: The behavior of GA_2 as R varies.

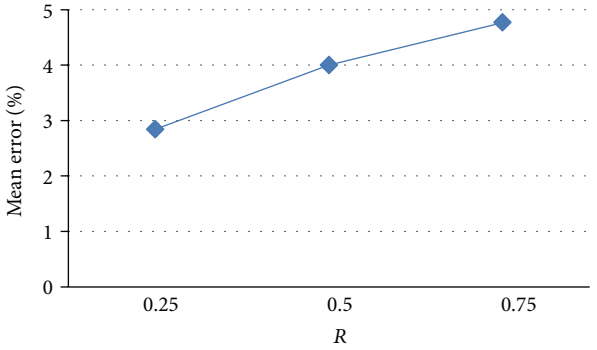

Figure 27: The behavior of GA_3 as R varies.

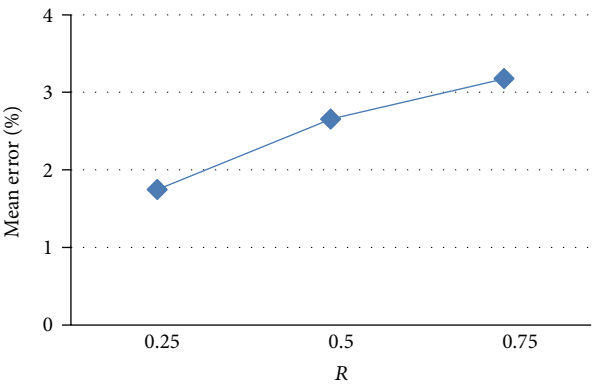

Figure 28: The behavior of GA^* as R varies.

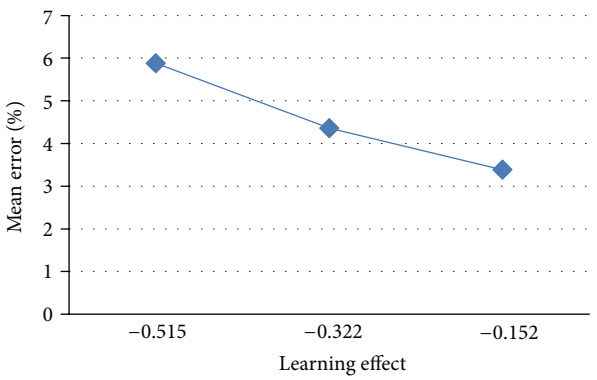

Figure 29: The behavior of GA_1 as a varies.

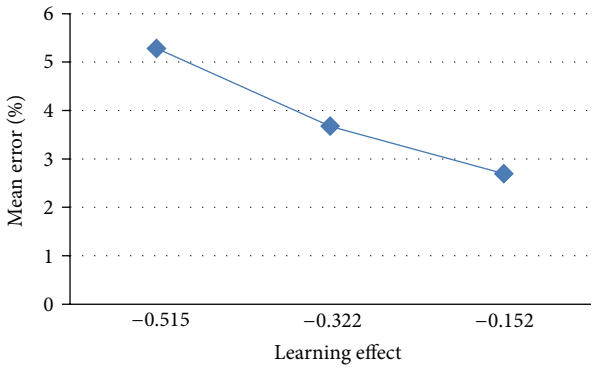

Figure 30: The behavior of GA_2 as a varies.

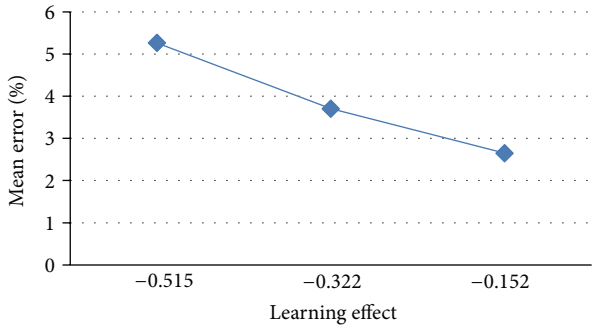

Figure 31: The behavior of GA_3 as a varies.

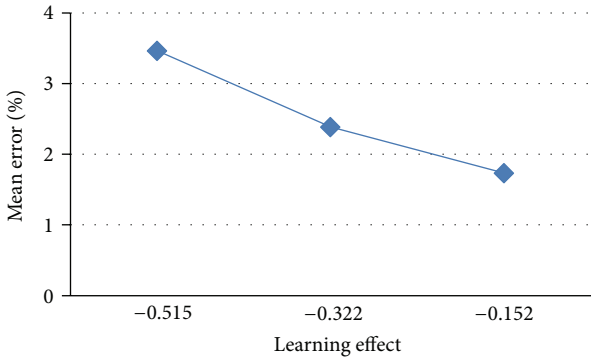

Figure 32: The behavior of GA^* as a varies.

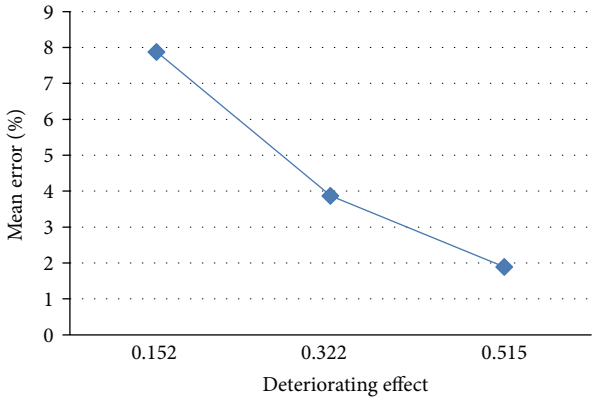

Figure 33: The behavior of GA_1 as β varies.

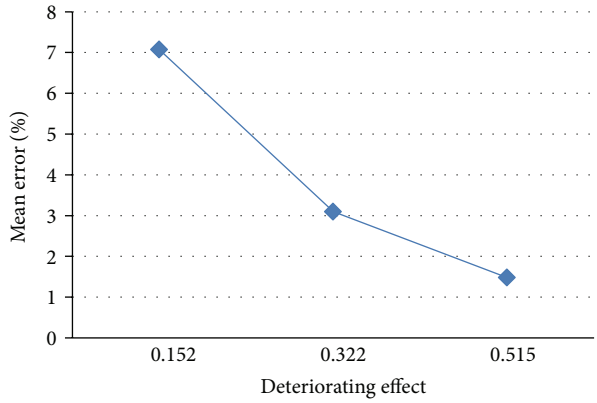

Figure 34: The behavior of GA_2 as β varies.

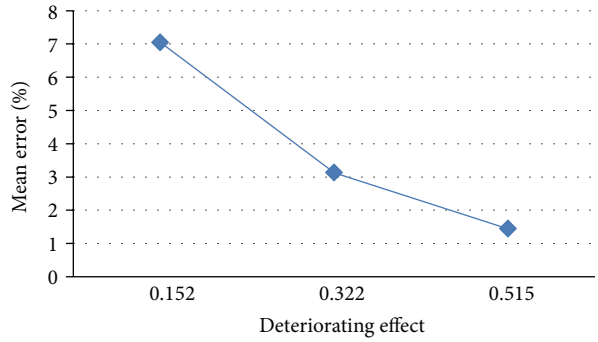

Figure 35: The behavior of GA_3 as β varies.

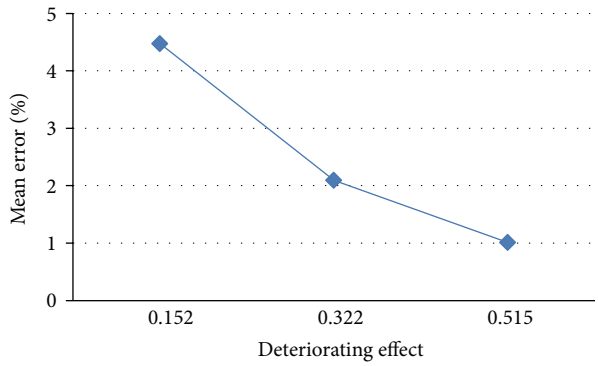

Figure 36: The behavior of GA^* as β varies.

pro = 0.5 in the experiments. We set the learning effect at 70%, 80%, and 90% (corresponding to $\alpha = -0.515$, -0.322, and -0.152, resp.) and the values of the deteriorating effect at 70%, 80%, and 90% (corresponding to $\beta = 0.515$, 0.322, and 0.152, resp.). We randomly generated a set of 100 instances for each situation. As a result, we examined 270 experimental situations. For each GA heuristic, we calculate its relative percentage deviation as

$$\text{RPD} = \frac{\text{GA}_i - \text{GA}^*}{\text{GA}^*} \times 100\%, \tag{17}$$

where GA_i is the objective function value generated by the GA heuristic and $\text{GA}^* = \min\{\text{GA}_i, i = 1, 2, 3\}$ is the GA

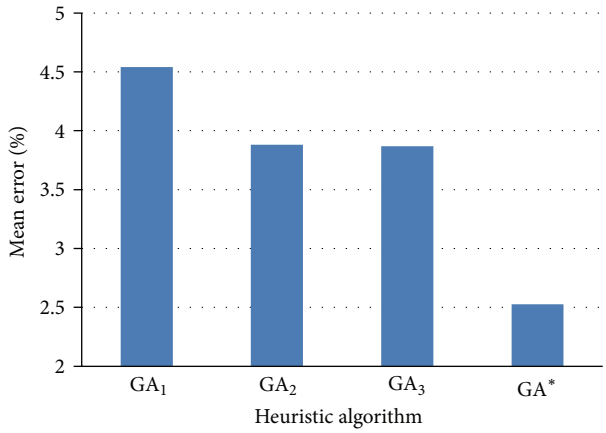

Figure 37: The behavior of GAs at $n = 14$.

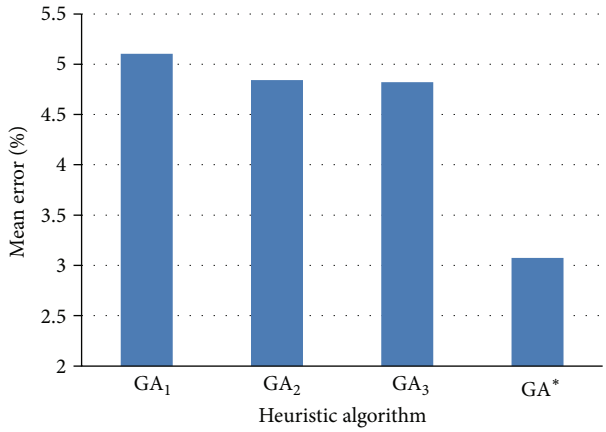

Figure 38: The behavior of GAs at $n = 16$.

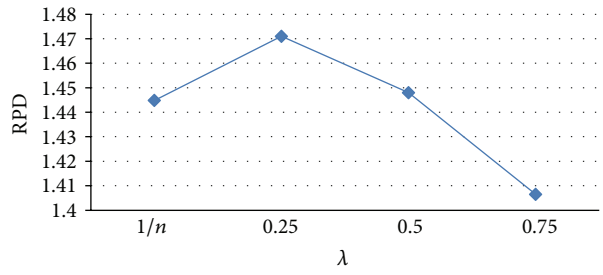

Figure 39: The behavior of GA_1 as λ varies.

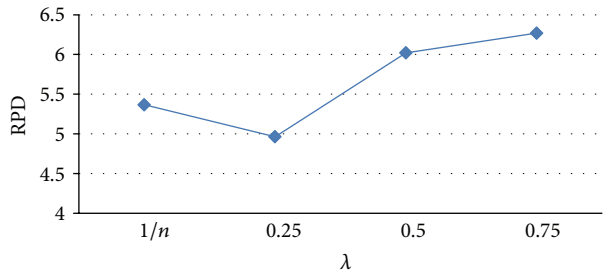

Figure 40: The behavior of GA_2 as λ varies.

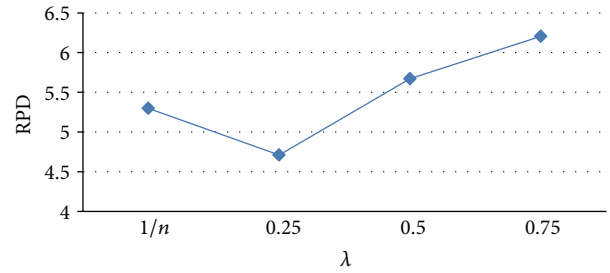

Figure 41: The behavior of GA_3 as λ varies.

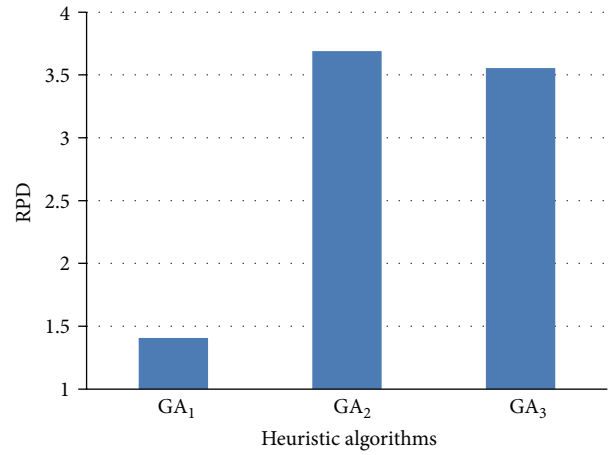

Figure 42: The behavior of GAs at $n = 30$.

heuristic that yields the smallest objective function value among the three GA algorithms. We recorded the average and maximum RPD, and the mean execution time for each heuristic. The results are summarized in the following figures.

As shown in Figures 39, 40, and 41, we observe that the RPD mean of GA_1 is in general smaller than those of GA_2 and GA_3. The effects of a, β, R, and τ are similar to those with smaller numbers of jobs.

Overall, we observe from Figures 42 and 43 that the grand mean of the RPD of GA_1 is smaller than that of GA_2 and GA_3. The result also shows that GA_2 and GA_3 slightly outperform GA_1 with smaller numbers of jobs, but the result is reversed with larger numbers of jobs. This implies that there is no absolute dominance relationship among three GAs. Thus, we recommend that the GA^* be used in order to attain stability and good quality solutions.

5. Conclusions

In this paper we study a two-agent single-machine scheduling problem with simultaneous considerations of deteriorating jobs, learning effects, and ready times. To search for optimal and near-optimal solutions, we propose a branch-and-bound algorithm incorporated with some dominance rules and a lower bound and three genetic algorithms, respectively.

The computational results show that our proposed branch-and-bound algorithm can solve instances with up to 16 jobs with reasonable numbers of nodes and execution

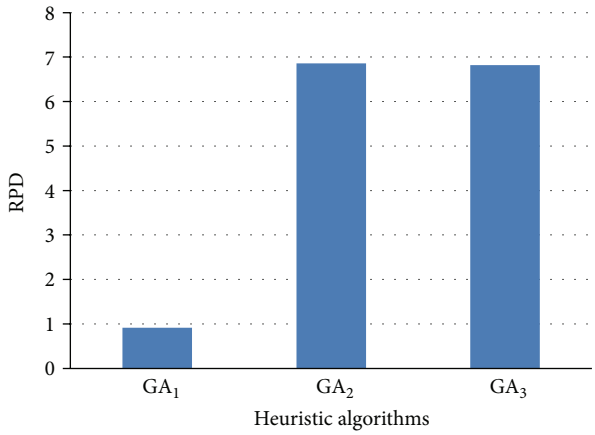

FIGURE 43: The behavior of GAs at $n = 40$.

times. In addition, the computational experiments reveal that the proposed GA* does well in terms of efficiency and solution quality. Future research may consider other scheduling criteria or study the problem in the multimachine setting.

Acknowledgments

The authors would like to thank the Editor and three anonymous referees for their helpful comments on an earlier version of the paper.

References

[1] S. Browne and U. Yechiali, "Scheduling deteriorating jobs on a single processor," *Operations Research*, vol. 38, no. 3, pp. 495–498, 1990.

[2] D. Biskup, "Single-machine scheduling with learning considerations," *European Journal of Operational Research*, vol. 115, no. 1, pp. 173–178, 1999.

[3] J. N. D. Gupta and S. K. Gupta, "Single facility scheduling with nonlinear processing times," *Computers and Industrial Engineering*, vol. 14, no. 4, pp. 387–393, 1988.

[4] B. Alidaee and N. K. Womer, "Scheduling with time dependent processing times: review and extensions," *Journal of the Operational Research Society*, vol. 50, no. 7, pp. 711–720, 1999.

[5] T. C. E. Cheng, Q. Ding, and B. M. T. Lin, "A concise survey of scheduling with time-dependent processing times," *European Journal of Operational Research*, vol. 152, no. 1, pp. 1–13, 2004.

[6] D. Biskup, "A state-of-the-art review on scheduling with learning effects," *European Journal of Operational Research*, vol. 188, no. 2, pp. 315–329, 2008.

[7] J.-B. Wang, "Single-machine scheduling problems with the effects of learning and deterioration," *Omega*, vol. 35, no. 4, pp. 397–402, 2007.

[8] J.-B. Wang and T. C. E. Cheng, "Scheduling problems with the effects of deterioration and learning," *Asia-Pacific Journal of Operational Research*, vol. 24, no. 2, pp. 245–261, 2007.

[9] X. Wang and T. C. Edwin Cheng, "Single-machine scheduling with deteriorating jobs and learning effects to minimize the makespan," *European Journal of Operational Research*, vol. 178, no. 1, pp. 57–70, 2007.

[10] J.-B. Wang and L.-L. Liu, "Two-machine flow shop problem with effects of deterioration and learning," *Computers and Industrial Engineering*, vol. 57, no. 3, pp. 1114–1121, 2009.

[11] J.-B. Wang and Q. Guo, "A due-date assignment problem with learning effect and deteriorating jobs," *Applied Mathematical Modelling*, vol. 34, no. 2, pp. 309–313, 2010.

[12] X. Zhang and G. Yan, "Single-machine group scheduling problems with deteriorated and learning effect," *Applied Mathematics and Computation*, vol. 216, no. 4, pp. 1259–1266, 2010.

[13] S.-J. Yang, "Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration," *Applied Mathematics and Computation*, vol. 217, no. 7, pp. 3321–3329, 2010.

[14] Z. Zhu, L. Sun, F. Chu, and M. Liu, "Due-window assignment and scheduling with multiple rate-modifying activities under the effects of deterioration and learning," *Mathematical Problems in Engineering*, vol. 2011, Article ID 151563, 19 pages, 2011.

[15] P. Liu and L. Tang, "Two-agent scheduling with linear deteriorating jobs on a single machine," *Lecture Notes in Computer Science*, vol. 5092, pp. 642–650, 2008.

[16] P. Liu, X. Zhou, and L. Tang, "Two-agent single-machine scheduling with position-dependent processing times," *International Journal of Advanced Manufacturing Technology*, vol. 48, no. 1–4, pp. 325–331, 2010.

[17] T. C. E. Cheng, W.-H. Wu, S.-R. Cheng, and C.-C. Wu, "Two-agent scheduling with position-based deteriorating jobs and learning effects," *Applied Mathematics and Computation*, vol. 217, no. 21, pp. 8804–8824, 2011.

[18] W.-H. Wu, S.-R. Cheng, C.-C. Wu, and Y. Yin, "Ant colony algorithms for a two-agent scheduling with sum-of processing times-based learning and deteriorating considerations," *Journal of Intelligent Manufacturing*, vol. 23, no. 5, pp. 1985–1993, 2012.

[19] C.-C. Wu, P.-H. Hsu, J.-C. Chen, and N.-S. Wang, "Genetic algorithm for minimizing the total weighted completion time scheduling problem with learning and release times," *Computers and Operations Research*, vol. 38, no. 7, pp. 1025–1034, 2011.

[20] Y. H. V. Lun, K. H. Lai, C. T. Ng, C. W. Y. Wong, and T. C. E. Cheng, "Research in shipping and transport logistics," *International Journal of Shipping and Transport Logistics*, vol. 3, pp. 1–5, 2011.

[21] F. Zhang, C. T. Ng, G. Tang, T. C. E. Cheng, and Y. H. V. Lun, "Inverse scheduling: applications in shipping," *International Journal of Shipping and Transport Logistics*, vol. 3, no. 3, pp. 312–322, 2011.

[22] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici, "Scheduling problems with two competing agents," *Operations Research*, vol. 52, no. 2, pp. 229–242, 2004.

[23] K. R. Baker and J. C. Smith, "A multiple-criterion model for machine scheduling," *Journal of Scheduling*, vol. 6, no. 1, pp. 7–16, 2003.

[24] J. J. Yuan, W. P. Shang, and Q. Feng, "A note on the scheduling with two families of jobs," *Journal of Scheduling*, vol. 8, no. 6, pp. 537–542, 2005.

[25] T. C. E. Cheng, C. T. Ng, and J. J. Yuan, "Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs," *Theoretical Computer Science*, vol. 362, no. 1–3, pp. 273–281, 2006.

[26] T. C. E. Cheng, C. T. Ng, and J. J. Yuan, "Multi-agent scheduling on a single machine with max-form criteria," *European Journal of Operational Research*, vol. 188, no. 2, pp. 603–609, 2008.

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

[28] A. Agnetis, D. Pacciarelli, and A. Pacifici, "Multi-agent single machine scheduling," *Annals of Operations Research*, vol. 150, no. 1, pp. 3–15, 2007.

[29] B. Mor and G. Mosheiov, "Scheduling problems with two competing agents to minimize minmax and minsum earliness measures," *European Journal of Operational Research*, vol. 206, no. 3, pp. 540–546, 2010.

[30] B. Mor and G. Mosheiov, "Single machine batch scheduling with two competing agents to minimize total flowtime," *European Journal of Operational Research*, vol. 215, no. 3, pp. 524–531, 2011.

[31] Y. Yin, W.-H. Wu, S.-R. Cheng, and C.-C. Wu, "An investigation on a two-agent single-machine scheduling problem with unequal release dates," *Computers and Operations Research*, vol. 39, no. 12, pp. 3062–3073, 2012.

[32] Y. Yin, S. R. Cheng, T. C. E. Cheng, W. H. Wu, and C. C. Wu, "Two-agent single-machine scheduling with release times and deadlines," *International Journal of Shipping and Transport Logistics*, vol. 5, pp. 76–94, 2013.

[33] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[34] S. K. Iyer and B. Saxena, "Improved genetic algorithm for the permutation flowshop scheduling problem," *Computers and Operations Research*, vol. 31, no. 4, pp. 593–606, 2004.

[35] I. Essafi, Y. Mati, and S. Dauzère-Pérès, "A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem," *Computers and Operations Research*, vol. 35, no. 8, pp. 2599–2616, 2008.

[36] O. Etiler, B. Toklu, M. Atak, and J. Wilson, "A genetic algorithm for flow shop scheduling problems," *Journal of the Operational Research Society*, vol. 55, no. 8, pp. 830–835, 2004.

[37] C. Reeves, "Heuristics for scheduling a single machine subject to unequal job release times," *European Journal of Operational Research*, vol. 80, no. 2, pp. 397–403, 1995.

[38] C.-L. Chen, V. S. Vempati, and N. Aljaber, "An application of genetic algorithms for flow shop problems," *European Journal of Operational Research*, vol. 80, no. 2, pp. 389–396, 1995.

[39] M. L. Fisher, "A dual algorithm for the one-machine scheduling problem," *Mathematical Programming*, vol. 11, no. 1, pp. 229–251, 1976.