Applied Soft Computing 12 (2012) 3580–3589

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Genetic algorithms for a two-agent single-machine problem with release time

Wen-Chiung Lee [a],*, Yu-Hsiang Chung [b], Mei-Chia Hu [a]

[a] Department of Statistics, Feng Chia University, Taichung, Taiwan
[b] Department of Industrial & Engineering Management, National Chiao Tung University, Hsinchu, Taiwan

ARTICLE INFO

Article history:
Received 6 February 2012
Received in revised form 11 May 2012
Accepted 11 June 2012
Available online 6 July 2012

Keywords:
Scheduling
Total tardiness
Two-agent
Single-machine
Release time
Maximum tardiness

ABSTRACT

Scheduling with two competing agents has drawn a lot of attention lately. However, it is assumed that all the jobs are available in the beginning in most of the research. In this paper, we study a single-machine problem in which jobs have different release times. The objective is to minimize the total tardiness of jobs from the first agent given that the maximum tardiness of jobs from the second agent does not exceed an upper bound. Three genetic algorithms are proposed to obtain the near-optimal solutions. Computational results show that the branch-and-bound algorithm could solve most of the problems with 16 jobs within a reasonable amount of time. In addition, it shows that the performance of the combined genetic algorithm is very good with mean error percentages of less than 0.2% for all the cases.

1. Introduction

Recently, there is a growing interest in multi-agent scheduling where jobs might come from several customers who have their own objective functions. For example, Baker and Smith [1] gave an example of a prototype shop where the manufacturing department might be concerned about finishing jobs before their due dates, and the research and development department might be more concerned about quick response time. Kubzin and Strusevich [2] presented another example in which the maintenance activities compete with real jobs for machine occupancy in maintenance planning. Meiners and Torng [3] gave a telecommunication example where various types of packets and service compete for the radio resource usage. Soomer and Franx [4] gave a transportation example where the agents own their transportation resources, and compete for the usage of the infrastructures. Leung et al. [5] pointed out that several important classes of scheduling problems, such as rescheduling problems or scheduling with availability constraints, can be formulated as two-agent scheduling problems. Baker and Smith [1] and Agnetis et al. [6] pioneered the scheduling problems with two competing agents. Since then, two-agent scheduling has drawn researchers' attention [7–16].

Recently, Leung et al. [5] generalized the single machine problems of Agnetis et al. [6] to the case of multiple identical parallel machines where job preemption is allowed. They also considered certain single-machine problems where the jobs may have different release dates, and job preemptions may or may not be allowed. Lee et al. [17] considered a two-agent scheduling problem on a two-machine permutation flowshop. Their objective is to minimize the total tardiness of jobs from the first agent given that the number of tardy jobs of the second agent is zero. Liu et al. [18] brought the aging and learning effects into the two-agent scheduling. Their objective is to minimize the total completion time of jobs from the first agent given that the maximum cost of jobs from the second agent cannot exceed a given upper bound. Wan et al. [19] considered several two-agent scheduling problems with controllable job processing times in which two agents have to share either a single machine or two identical machines in parallel while processing their jobs. Mor and Mosheiov [20] considered a two-agent scheduling problem on a single-machine problem to minimize the maximum earliness cost or total (weighted) earliness cost of jobs from one agent, subject to an upper bound on the maximum earliness cost of jobs from the other agent. They introduced a polynomial-time solution for the maximum earliness problem and proved NP-hardness for the weighted earliness case. Lee et al. [21] considered a two-agent problem where the objective is to minimize the total completion time of jobs from the first agent given that no tardy job is allowed for the second agent. Liu et al. [22] developed the optimal solutions for certain two-agent problems with increasing linear deterioration on a single machine. Their goal is to minimize the objective function of the first agent given that the objective function of the second agent cannot exceed a given bound. Nong et al. [23] considered a two

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

1568-4946/$ – see front matter
http://dx.doi.org/10.1016/j.asoc.2012.06.015

agent problem on a single machine where the objective is to minimize the weighted sum of the maximum completion time of jobs from one agent and the total weighted completion time of jobs from the other agent. They provided a 2-approximation algorithm and showed the case is NP-hard when the number of jobs of the first agent is fixed. Yin et al. [24] studied three single-machine problems with deteriorating jobs. The objectives are the maximum earliness cost, total earliness cost, and total weighted earliness cost, while keeping the maximum earliness cost of jobs from the other agent below a fixed level. Mor and Mosheiov [25] considered a single-machine problem with batch scheduling to minimize the total completion time of jobs from one agent, given that the maximum completion time of jobs from the other agent does not exceed an upper bound. Wu et al. [26] studied single-machine scheduling with learning effects. Their objective is to minimize the total tardiness of jobs from the first agent, given that no tardy job is allowed for the second agent. Cheng et al. [27] considered single-machine scheduling with truncated learning effects. Their objective is to minimize the total weighted completion time of jobs from the first agent, given that no tardy job is allowed for the second agent. Li and Hsu [28] investigated a single-machine problem with learning effect where the objective is to minimize the total weighted completion time of both agents with the restriction that the makespan of either agent cannot exceed an upper bound.

Most of the research in scheduling with two competing agents assumes that jobs are ready to be processed in the beginning. However, customer orders might not arrive simultaneously in many realistic situations. Thus, it is more practical to consider jobs release times. Leung et al. [5] were the only authors who considered two-agent scheduling with job release times. In this paper, we study a two-agent scheduling problem on a single machine with release time where the objective is to minimize the total tardiness of jobs from the first agent given that the maximum tardiness of jobs from the second agent cannot exceed an upper bound. To the best of our knowledge, this problem has never been studied. The rest of this paper is organized as follows. In the next section, the formulation of our problem is described. In Section 3, a branch-and-bound algorithm with several elimination rules and a lower bound is developed. In Section 4, three genetic algorithms are proposed to solve this problem. In Section 5, computational experiments are conducted to evaluate the performance of the genetic algorithms. A conclusion is given in the final section.

2. Problem description

The problem formulation is described as follows. There are n jobs, each belongs to either agent AG_1 or AG_2. For each job j, there is a processing time p_j, a due date d_j, a release time r_j, and an agent code I_j, where $I_j = 1$ if $j \in AG_1$ or $I_j = 2$ if $j \in AG_2$. Under a schedule S, let $C_j(S)$ be the completion time of job j and let $T_j(S) = \max\{0, C_j(S) - d_j\}$ be the tardiness of job j. In this paper, we consider a single machine problem to minimize the total tardiness of jobs from agent AG_1 given that the maximum tardiness of jobs from agent AG_2 does not exceed an upper bound M. Using the three-field notation extended by Agnetis et al. [6], this problem is denoted by $1|r_j^1; r_j^2|\sum T_j; T_{\max}$.

3. A branch-and-bound algorithm

When all the jobs are from agent AG_1 and the release times are zero, the problem reduces to the classical single-machine total tardiness time problem which is NP-hard [29]. Therefore, a branch-and-bound algorithm is proposed to derive the optimal solution.

3.1. Dominance properties

First, we provide a result to speed up the search process. We then develop several adjacent dominance properties to reduce the searching scope.

Theorem 1. *If there is a job i such that $r_i + p_i \le r_j$ for all the remaining jobs j, then job i is scheduled first in the optimal sequence.*

Proof. The proof is omitted since it is straightforward.

Suppose that S and S' are two schedules of jobs with the only difference between them a pairwise interchange of two adjacent jobs i and j. That is, $S = (\pi, i, j, \pi')$ and $S' = (\pi, j, i, \pi')$, where π and π' each denote a partial sequence. In addition, let t be the completion time of the last job in π. The completion times of jobs i and j in S are

$$C_i(S) = \max\{t, r_i\} + p_i \tag{1}$$

and

$$C_j(S) = \max\{C_i(S), r_j\} + p_j \tag{2}$$

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

$$C_j(S') = \max\{t, r_j\} + p_j \tag{3}$$

and

$$C_i(S') = \max\{C_j(S'), r_i\} + p_i \tag{4}$$

Depending on whether jobs are from agents AG_1 or AG_2, we divide the situation into the following three cases.

Case 1. Both jobs i and j are from agent AG_1.

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

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

Proof. Since $t \ge \max\{r_i, r_j\}$, we have

$$C_i(S) = t + p_i$$

$$C_j(S) = t + p_i + p_j$$

$$C_j(S') = t + p_j$$

and

$$C_i(S') = t + p_j + p_i$$

Therefore, we have $C_j(S) \le C_i(S')$. Since $t + p_i \ge d_i$, we have

$$T_i(S) = t + p_i - d_i \tag{5}$$

and

$$T_i(S') = t + p_j + p_i - d_i \tag{6}$$

Suppose that $T_i(S)$ is not zero. Note that this is the more restrictive case since it comprises the case that $T_i(S)$ is zero. From Eqs. (5) and (6), we have

$$T_j(S') + T_i(S') - T_i(S) - T_j(S) = d_j - t - p_i > 0$$

since $t + p_i < d_j$. Thus, S dominates S'.

Property 1.2. *If $t \ge \max\{r_i, r_j\}$ and $d_i < t + p_i + p_j \le d_j$, then S dominates S'.*

Property 1.3. *If $t \ge \max\{r_i, r_j\}$, $t + p_i \le d_i \le t + p_j + p_i$, and $d_j > d_i$, then S dominates S'.*

Property 1.4. *If $r_i \le t \le r_j \le t+p_i$, $d_j \ge t+p_i+p_j$, and $d_i < r_j+p_j+p_i$, then S dominates S′.*

Property 1.5. *If $r_i \le t \le r_j \le t+p_i$, $t+p_i \le d_i \le r_j+p_j+p_i$, and $d_j > d_i$, then S dominates S′.*

Property 1.6. *If $r_i \le t \le r_j \le t+p_i$ and $d_i \le t+p_i \le d_j$, then S dominates S′.*

Property 1.7. *If $t \le r_i \le r_j \le r_i+p_i$, $r_i+p_i+p_j \le d_j$, and $r_j+p_j+p_i > d_i$, then S dominates S′.*

Property 1.8. *If $t \le r_i \le r_j \le r_i+p_i \le d_i \le r_j+p_j+p_i$ and $r_i+d_i < r_j+d_j$, then S dominates S′.*

Property 1.9. *If $t \le r_i \le r_j \le r_i+p_i < d_j$ and $r_i+p_i \ge d_i$, then S dominates S′.*

Property 1.10. *If $\max\{t, r_i\}+p_i \le r_j$ and $r_j+p_j+p_i > d_i$, then S dominates S′.*

Case 2. Job i is from agent AG_1, but job j is from agent AG_2.

To show that S dominates S', it suffices to show that $T_j(S) \le M$, $T_i(S) < T_i(S')$ and $C_j(S) - C_i(S') \le 0$.

Property 2.1. *If $t \ge \max\{r_i, r_j\}$ and $t+p_i+p_j-d_j \le M$, then S dominates S′.*

Property 2.2. *If $r_i \le t \le r_j \le t+p_i$ and $t+p_i+p_j-d_j \le M$, then S dominates S′.*

Property 2.3. *If $t \le r_i \le r_j \le r_i+p_i$ and $r_i+p_i+p_j-d_j \le M$, then S dominates S′.*

Property 2.4. *If $t \ge r_i$, $t+p_i \le r_j$, and $r_j+p_j-d_j \le M$, then S dominates S′.*

Property 2.5. *If $t \le r_i$, $r_i+p_i \le r_j$, and $r_j+p_j-d_j \le M$, then S dominates S′.*

Case 3. Both jobs i and j are from agent AG_2.

To show that S dominates S', it suffices to show that $T_i(S) \le M$, $T_j(S) \le M$ and $C_j(S) - C_i(S') < 0$.

Property 3.1. *If $t \ge \max\{r_i, r_j\}$, $t+p_i-d_i \le M$, $t+p_i+p_j-d_j \le M$, and $d_i < d_j$, then S dominates S′.*

Property 3.2. *If $r_i \le t < r_j \le t+p_i$, $t+p_i-d_i \le M$, and $t+p_i+p_j-d_j \le M$, then S dominates S′.*

Property 3.3. *If $t \le r_i < r_j \le r_i+p_i$, $r_i+p_i-d_i \le M$, and $r_i+p_i+p_j-d_j \le M$, then S dominates S′.*

Property 3.4. *If $t \ge r_i$, $t+p_i \le r_j$, $t+p_i-d_i \le M$, and $r_j+p_j-d_j \le M$, then S dominates S′.*

Property 3.5. *If $t \le r_i$, $r_i+p_i \le r_j$, $r_i+p_i-d_i \le M$, and $r_j+p_j-d_j \le M$, then S dominates S′.*

To further facilitate the search process, we provide a proposition to determine the feasibility of a partial schedule. Assume that (π, π^c) is a sequence of jobs where π is the scheduled part and π^c is the unscheduled part.

Proposition 1. If there is a job $j \in \pi^c \cap AG_2$ such that $t+p_j > d_j+M$, then (π, π^c) is not a feasible sequence.

3.2. A lower bound

In this subsection we develop a lower bound for the branch-and-bound algorithm. Let *PS* be a partial sequence in which s jobs are scheduled. Suppose that, among the unscheduled set *US* with $n-s$ jobs, there are n_1 jobs from agent AG_1 and n_2 jobs from agent AG_2, where $n_1+n_2=n-s$. For these unscheduled jobs, we have $p_{(s+1)} \le p_{(s+2)} \le \cdots \le p_{(n)}$ when they are arranged in non-decreasing order of their processing times and $r_{(s+1)} \le r_{(s+2)} \le \cdots \le r_{(n)}$ when they are arranged in the non-decreasing order of their release times. Note that $p_{(i)}$ and $r_{(i)}$ may not be from the same job. Furthermore, the due dates of the n_1 (n_2) unscheduled jobs from agent AG_1 (AG_2) are denoted as $d^1_{(1)} \le d^1_{(2)} \le \cdots \le d^1_{(n_1)}$ ($d^2_{(1)} \le d^2_{(2)} \le \cdots \le d^2_{(n_2)}$) when they are in non-decreasing order of their due dates. The idea of the proposed lower bound is that we first derive a lower bound on the completion times of the unscheduled jobs based on the SPT rule, and then we assign them to agents AG_1 and AG_2 without violating the constraint that the maximum tardiness of jobs from agent AG_2 does not exceed the upper bound M. In the first step, the completion time of the $(s+1)$th job is

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

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

$$C_{[s+i]} \ge C_{[s]} + \sum_{l=1}^{i} p_{(s+l)} \tag{7}$$

On the other hand, this lower bound might not be tight if the release times are large. Thus, $C_{[s+1]} = \max\{C_{[s]}, r_{[s+1]}\} + p_{[s+1]} \ge r_{(s+1)} + p_{(s+1)}$.

By induction, we have

$$C_{[s+i]} = \max_{1 \le k \le i}\left\{ r_{[s+k]} + \sum_{l=1}^{i-k+1} p_{[s+k+l]} \right\} \ge \max_{1 \le k \le i}\left\{ r_{(s+k)} + \sum_{l=1}^{i-k+1} p_{(s+l)} \right\} \tag{8}$$

From Eqs. (7) and (8), a lower bound on the completion time of the $(s+i)$th job is

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

for $i=1, 2, \ldots, n-s$. In the second step, the remaining task is to assign the estimated completion times to the jobs from agent AG_1 or AG_2. The principle is to assign the completion times to the jobs from agent AG_2 as late as possible without violating the assumption that the maximum tardiness of the jobs of agent AG_2 cannot exceed the upper bound. In addition, let $C^1_{(1)} \le C^1_{(2)} \le \cdots \le C^1_{(n_1)}$ and $C^2_{(1)} \le C^2_{(2)} \le \cdots \le C^2_{(n_2)}$ denote the estimated completion times of the jobs from agents AG_1 and AG_2, respectively, when they are arranged in non-decreasing order. The assignment procedure is in a backward manner starting from the job with the remaining largest due date until all the jobs are assigned. The details are given as follows:

Algorithm of the lower bound:

Step 1: Set $ic=n-s$, $i_1=n_1$, $i_2=n_2$, and $C_{(s+i)} = \max\{t + \sum_{l=1}^{i} p_{(s+l)}, \max_{1 \le k \le i}\{r_{(s+k)} + \sum_{l=1}^{i-k+1} p_{(s+l)}\}\}$ for $i=1, 2, \ldots, n-s$.
Step 2: If $C_{(s+ic)} \le d^2_{(i_2)} + M$, then set $C^2_{(i_2)} = C_{(s+ic)}$ and $i_2 = i_2 - 1$. Otherwise, set $C^1_{(i_1)} = C_{(s+ic)}$ and $i_1 = i_1 - 1$.
Step 3: Set $ic = ic - 1$. If $ic \ge 1$, then go to Step 2.

Therefore, a lower bound on the total tardiness of jobs from agent AG_1 for *PS* is

$$LB = \sum_{j \in AG_1} T_j(PS) + \sum_{j=1}^{n_1} \max\{0, C^1_{(j)} - d^1_{(j)}\}$$

3.3. Description of the branch-and-bound algorithms

A depth-first search is used in the branching procedure starting from the first position. We choose a branch and systematically work down the tree until we either eliminate it or reach its final node, in which case this sequence either replace the initial solution or is eliminated. The outline of the branch-and-bound algorithm is as follows.

Step 1. {Initialization} Implement the genetic algorithms (discussed in the next section) to obtain a sequence as the initial incumbent solution.
Step 2. {Branching} Apply Theorem 1, Properties 1.1 to 3.5, and Proposition 1 to eliminate the dominated partial sequence.
Step 3. {Bounding} For the non-dominated nodes, compute the lower bound of the total tardiness of jobs from agent AG_1 of the unfathomed partial sequences or that of the completed sequences. If the lower bound on the objective function for the partial sequence is greater than the initial solution, eliminate that node and all the nodes beyond it in the branch. If the objective function of the completed sequence is less than the initial solution, replace it as the new solution. Otherwise, eliminate it.

4. Genetic algorithms

Evolutionary algorithms have become popular in obtaining good approximate solutions for many NP-hard problems [30–35]. In this paper, we utilize the genetic algorithm (*GA*). It is an intelligent random search strategy which has been used successfully to find near optimal solutions to many complex problems [36–38]. The *GA* usually starts with a population of feasible solutions and iteratively replaces the current population by a new population until certain stopping condition is reached. It requires a suitable encoding for the problem and a fitness function that represents a measure of the quality of each encoded solution (chromosome). The reproduction mechanism selects the parents and recombines them using a crossover operator to generate offspring which are submitted to a mutation operator in order to alter them locally to avoid premature convergence. The components of the *GA* applied to our problem are as follows.

4.1. Encoding

In this study, we adopt the random number encoding method [39]. For a problem of n jobs, we generate a chromosome with n uniform random real numbers between 0 and 1 to represent the genes, where each gene corresponds to a job. The order of these random numbers represents the job sequence. For instance, the chromosome of a 5-job problem (0.33, 0.78, 0.13, 0.94, 0.26) would stand for the sequence (3, 5, 1, 2, 4).

4.2. Population size

The population size is an important factor in the performance of *GA*. For a large population size, it is easier to obtain a better solution, but it consumes more time. After a preliminary trial, the population size N is set at 500 in our computational experiment.

4.3. Fitness function

In order to mimic the natural process of the survival of the fittest, the fitness function assigns to each member of the population a value reflecting their relative superiority. In this paper, we adopt the idea by Homaifar et al. [40] of adding a penalty function to the

Parent1	0.45	0.32	0.15	0.78	0.53	0.36
Offspring	0.45	0.32	0.27	0.49	0.18	0.72
Parent2	0.18	0.87	0.27	0.49	0.18	0.72

Fig. 1. One cut-point crossover.

infeasible solution. Thus, the objective function of chromosome k is

$obj_k = \sum_{j \in AG_1} T_j + \alpha \max_{j \in AG_2} \max\{T_j - M, 0\}$, where α is set at 5000 in this study. In addition, we use the reciprocal of the objective value as the fitness value for each chromosome, and the probability that a chromosome is selected as the parent is proportional to its fitness value. That is, the probability of selecting chromosome i is $f_i = h_i / \sum_{j=1}^{N} h_j$, where $h_i = 1/\mathrm{obj}_i$, $i = 1, \ldots, N$, is the reciprocal of the objective value of chromosome i in a population of size N. This is to ensure that the probability of selection for a sequence with lower value of the objective function is higher.

4.4. Crossover

Crossover is an operation to generate new offspring from two parents. It is the main operator in GA. In this study, we use the one cut-point crossover as shown in Fig. 1 and the rate P_c was chosen at 95% after some pretests.

4.5. Mutation

Mutation is another main operator to prevent premature convergence and fall into local optimum. Such an operation can be viewed as a transition from a current solution to its neighborhood solution in a local search algorithm. In this study, we use the one-point mutation as shown in Fig. 2 and the mutation rate P_m is set at 80% based on our preliminary experiment.

4.6. Selection

It is a procedure to select offspring from parents to the next generation. In our study, the population size is fixed at 500 from generation to generation. In our study, we choose the best 50 chromosomes (10%) from the parent population and the best 450 chromosomes (90%) from the offspring to form the next generation.

4.7. Termination

After some pretests, we terminate the proposed *GA* after 20n generations, where n is the number of jobs.

4.8. Initial sequences

A good initial sequence might be useful to facilitate the convergence of the process or to obtain a better approximate solution. In this paper, three methods are implemented. In the first *GA* (GA_1),

Offspring	0.45	0.32	0.15	0.78	0.53	0.36
Offspring	0.45	0.32	0.38	0.78	0.53	0.36

Fig. 2. One-point mutation.

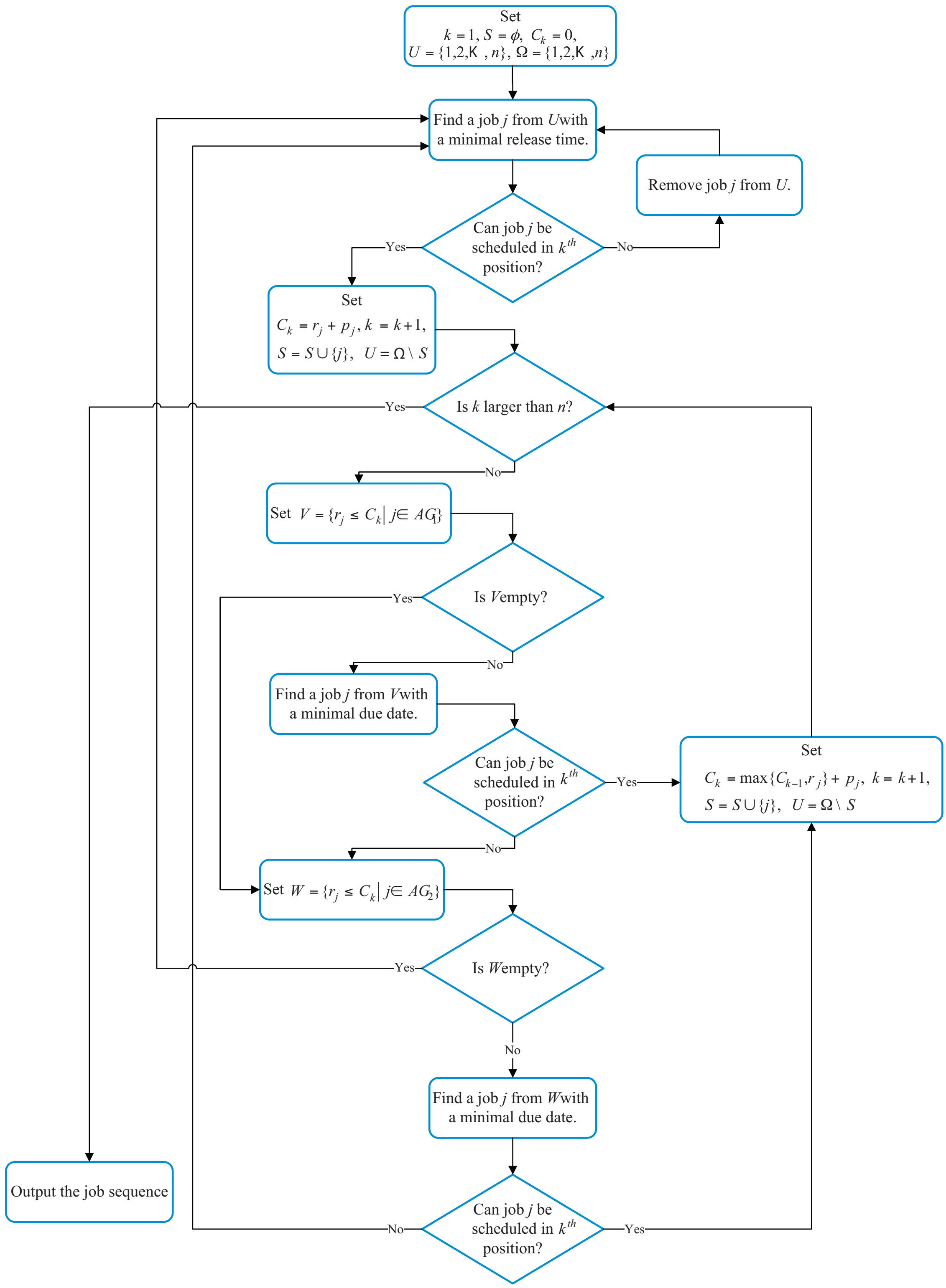

Fig. 3. Block diagram for *HA*.

the first generation consists of 500 random sequences. In the second *GA* (GA_2), the first generation consists of 499 random sequences and one designated sequence from the heuristic algorithm (*HA*) as described below. In the third *GA* (GA_3), the first generation consists of 51 random sequences and 441 designated sequences. The designated sequences sort jobs according to the non-decreasing order of $w_1 r_j + w_2 p_j + (1 - w_1 - w_2) d_j$, where $w_1 = 0, 1/60, \ldots, 20/60$ and $w_2 = 0, 1/60, \ldots, 20/60$. The algorithm is given below and shown in Fig. 3.

Heuristic algorithm (*HA*)

Step 1. Set $k = 1$, $S = \phi$, $U = \{1, \cdots, n\}$, $C_k = 0$, $\Omega = \{1, \cdots, n\}$.

Table 1
The performance of the branch-and-bound algorithm with $n=12$, $P=50\%$, $\lambda=1$, and $M=30n$.

τ	R	Number of nodes		CPU time	
		Mean	Max	Mean	Max
0.25	0.25	52.32	474	0.001	0.016
	0.50	71.59	1314	0.001	0.016
	0.75	46.37	328	0.001	0.016
0.50	0.25	117.22	1889	0.001	0.016
	0.50	121.95	776	0.001	0.016
	0.75	156.68	1870	0.002	0.016
0.75	0.25	107.12	583	0.001	0.016
	0.50	27.37	698	0.001	0.016

Step 2. Find a job j from U with a minimal release time.
Step 3. If job j can be scheduled in the kth position without causing the violation of the constraint, put job j in the kth position, set $C_k=r_j+p_j$, $k=k+1$, $S=S\cup\{j\}$, $U=\Omega\setminus S$. Otherwise, delete $\{j\}$ from U and go to Step 2.
Step 4. If $k>n$, go to Step 8. Otherwise, form the set $V=\{r_j\le C_k \text{ and } j\in AG_1\}$, and if V is empty, go to Step 6.
Step 5. Find a job j from V with a minimal due date. If job j can be scheduled in the kth position without causing the violation of the constraint, set job j in the kth position, set $C_k=\max\{C_{k-1}, r_j\}+p_j$, $k=k+1$, $S=S\cup\{j\}$, $U=\Omega\setminus S$, and go to Step 4.
Step 6. Form the set $W=\{r_j\le C_k \text{ and } j\in AG_2\}$. If W is empty, go to Step 2.
Step 7. Find a job j from W with a minimal due date. If job j can be scheduled in the kth position without causing the violation of the constraint, set job j in the kth position, $C_k=\max\{C_{k-1}, r_j\}+p_j$, $k=k+1$, $S=S\cup\{j\}$, $U=\Omega\setminus S$, and go to Step 4. Otherwise, go to Step 2.
Step 8. Output the job sequence.

4.9. Computational experiments

A computational experiment is conducted in this section to evaluate the performance of the branch-and-bound and the *GA*s. All the algorithms are coded in Fortran 90 and run on a personal computer with AMD Athlon(tm) 64 Processor 3500+, 2.21 GHz and 1 GB RAM under Windows XP. The processing times are generated from a uniform distribution over the integers 1–100. The job release times are generated from uniform distributions between 0 and $50.5n\lambda$ where n is the number of jobs and λ is a control variable, as suggested in [41]. The due date of job j is generated from a uniform distribution over the integers between $r_j+T(1-\tau-R/2)$ and $r_j+T(1-\tau+R/2)$, where r_j is the due date of job j, T is the total job processing times, τ is the tardiness factor, and R is the due date range. To ensure the feasibility of the instance, jobs from agent AG_2 are placed based on the EDD rule, and it is regenerated if the maximum tardiness of jobs from agent AG_2 exceeds the upper bound M.

The computational experiments are divided into four parts. The first part is to test the impact of the due date factors τ and R to the performance of the branch-and-bound algorithm. The number of jobs is 12, and P, the proportion of jobs from agent AG_1, is 50%. The release time factor λ is 1 and the upper bound of maximum tardiness is $30n$, where n is the number of jobs. Eight combinations of (τ, R) values are used, i.e. (0.25, 0.25), (0.25, 0.50), (0.25, 0.75), (0.5, 0.25), (0.5, 0.50), (0.5, 0.75), (0.75, 0.25), and (0.75, 0.50). The mean and maximum numbers of nodes and the mean and maximum CPU times (in seconds) are reported for the branch-and-bound algorithm. 100 instances are randomly generated for each case and the results are presented in Table 1 and Fig. 4. It is seen that the tardiness factor τ is more significant than the range factor R to the performance of the branch-and-bound algorithm. Problems are more difficult to solve when $\tau=0.5$. Moreover, the case $(\tau, R)=(0.5, 0.75)$ has the most mean number of nodes among the 8 cases.

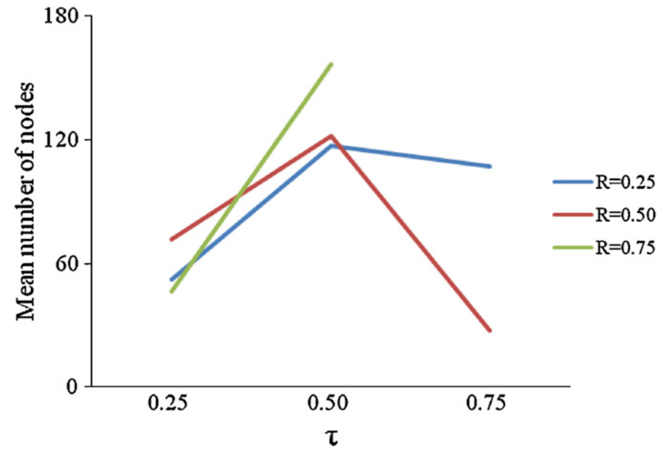

Fig. 4. The mean numbers of nodes of the branch-and-bound algorithm with $n=12$, $P=50\%$, $\lambda=1$, and $M=30n$.

The second part of the experiment is to test the impacts of the job release time (λ), the proportion of jobs from agent AG_1 (P), and the upper bound of the maximum tardiness (M) to the performance of the branch-and-bound algorithm. The number of jobs is 12, and the value of (τ, R) is (0.5, 0.5). Three values of λ (0.2, 1.0, 3.0), of P (0.25, 0.50, 0.75) and of M ($10n$, $30n$, $50n$, where n is the number of jobs) are tested. As a result, 27 cases are considered and 100 instances are randomly generated for each case. The results are presented in Table 2 and Figs. 5–7. It is seen that the job release time (λ) is the most significant factor among these three factors. In addition, problems are more difficult to solve when the value of λ is smaller. The proportion of jobs from agent AG_1 (P) is the second most significant factor, and problems tend to be harder when the value of P is smaller. On the other hand, the upper bound of the maximum tardiness (M) seems to have little influence on the performance of the branch-and-bound algorithm.

The third part of the experiment is to study the performance of the branch-and-bound algorithm and the accuracy of the three proposed genetic algorithms when the number of jobs is 16. We fix $\tau=0.5$ and $\lambda=0.2$ since problems are the most difficult to solve as shown in the results of the first and the second parts of the experiments. In addition, three different values of R (0.25, 0.5, 0.75), of P (0.25, 0.5, 0.75), and of M ($10n$, $30n$, $50n$) are chosen. The mean and the maximum numbers of nodes and the mean and the maximum

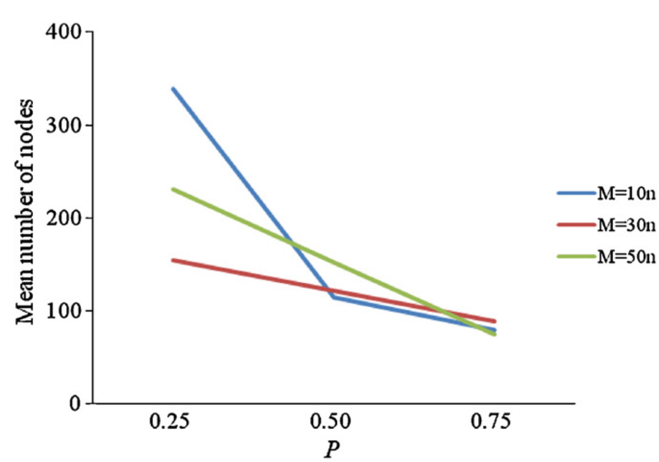

Fig. 5. The mean numbers of nodes of the branch-and-bound algorithm with $n=12$, $\lambda=1$, $\tau=0.5$, and $R=0.5$.

Table 2
The performance of the branch-and-bound algorithm with $n = 12$, $\tau = 0.5$, and $R = 0.5$.

λ	P	M	Number of nodes		CPU time	
			Mean	Max	Mean	Max
0.20	0.25	10n	7181.83	60,051	0.060	0.469
		30n	19,012.42	166,562	0.155	1.172
		50n	14,122.12	159,389	0.118	1.000
	0.50	10n	3253.14	21,118	0.025	0.156
		30n	1210.51	27,201	0.010	0.172
		50n	971.36	9635	0.008	0.063
	0.75	10n	932.56	9003	0.008	0.063
		30n	852.61	7611	0.005	0.063
		50n	889.96	12,724	0.005	0.078
1.00	0.25	10n	339.41	4462	0.002	0.031
		30n	154.73	2084	0.001	0.016
		50n	230.49	3733	0.001	0.016
	0.50	10n	114.85	1012	0.001	0.016
		30n	121.95	776	0.001	0.016
		50n	152.39	1790	0.001	0.016
	0.75	10n	79.14	531	0.001	0.016
		30n	89.12	736	0.001	0.016
		50n	74.93	350	0.001	0.016
3.00	0.25	10n	15.71	51	0.000	0.000
		30n	17.01	181	0.000	0.016
		50n	14.33	28	0.001	0.016
	0.50	10n	15.19	43	0.000	0.016
		30n	14.42	45	0.001	0.016
		50n	14.43	46	0.000	0.016
	0.75	10n	14.67	37	0.001	0.016
		30n	15.26	54	0.000	0.016
		50n	13.94	41	0.000	0.016

CPU times (in seconds) are reported for the branch-and-bound algorithm, while only the mean and the maximum error percentages of the *GA*s are given. For instance, the error percentage of the solution produced by GA_1 is calculated as

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

where V is the objective function of the sequence generated by GA_1 and V^* is the objective function of the optimal sequence from the branch-and-bound algorithm. For each case, 100 random instances are generated and the results are given in Table 3. Note that the branch-and-bound algorithm is terminated if the number of nodes explored is over 10^8, which was approximately 0.5 h in terms of the execution time. The instance with number of nodes over 10^8 is denoted as an asterisk in Table 3. It is observed that the branch-and-bound algorithm can solve most of the problems with 16 jobs in a reasonable amount of time. Among the 2700 problems, there are only 4 unsolvable problems. A closer look reveals that, among the three factors considered, the proportion of jobs from agent AG_1 (P) is the most significant one, and problems tend to be harder when P is smaller. The due date range (R) is the second significant, and problems are more difficult when R is smaller. As to the performance of *GA*s, it is noticed that the performance of all the three *GA*s is quite good. In addition, it is seen that *GA* with more designated initial sequences tends to have better overall solutions. However, there is an instance in which GA_1 yields an objective value of 3 but the total tardiness is 0 for the optimal sequence. Thus, we would

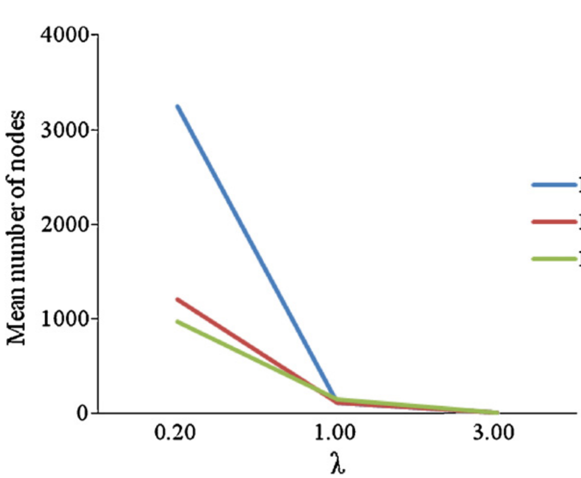

Fig. 6. The mean numbers of nodes of the branch-and-bound algorithm with $n = 12$, $P = 50\%$, $\tau = 0.5$, and $R = 0.5$.

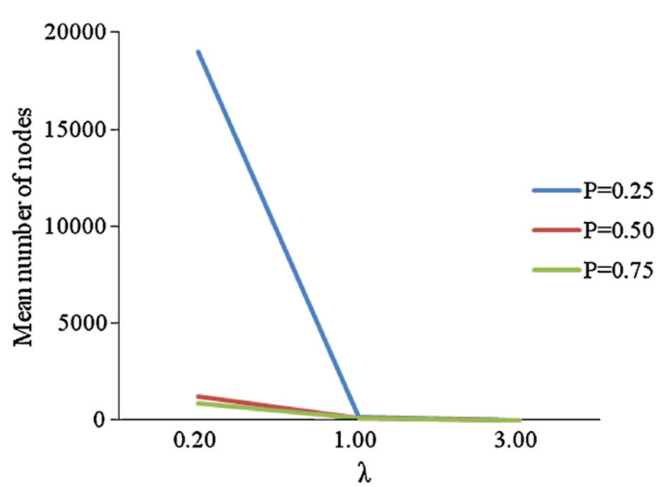

Fig. 7. The mean numbers of nodes of the branch-and-bound algorithm with $n = 12$, $M = 30n$, $\tau = 0.5$, and $R = 0.5$.

Table 3
The performance of the proposed algorithms with $n = 16$, $\lambda = 0.2$, and $\tau = 0.5$.

R	P	M	Branch-and-bound algorithm						Error percentages					
			Number of nodes		CPU time		GA_1		GA_2		GA_3		GA^*	
			Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
0.25	0.25	10n	7,733,551.02	88,667,901**	104.54	1256.59	0.10	4.41	0.06	2.54	0.06	2.54	0.06	2.54
		30n	7,419,256.32	86,013,946	103.43	1255.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	7,355,622.14	83,886,484*	109.69	1246.44	0.05	5.30	0.00	0.00	0.00	0.00	0.00	0.00
	0.50	10n	689,226.16	9,501,342	7.79	113.78	0.38	20.18	0.29	16.94	0.15	7.44	0.01	1.18
		30n	0.00	0	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	10,060.48	594,503	0.15	8.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.75	10n	137,298.25	1,636,596	1.63	18.66	0.84	83.66	0.84	83.66	0.13	12.99	0.00	0.00
		30n	0.00	0	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	0.00	0	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.50	0.25	10n	2,760,314.24	37,327,750	35.86	449.00	0.27	15.29	0.08	5.61	0.19	7.44	0.00	0.00
		30n	1,492,637.44	33,707,790	19.67	361.89	0.02	1.56	0.02	1.56	0.02	1.56	0.02	1.56
		50n	737,745.28	11,652,452	10.35	153.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.50	10n	281,446.60	4,129,379	3.44	51.52	0.53	36.40	0.55	36.40	0.21	16.07	0.16	16.07
		30n	3015.02	301,502	0.04	3.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	3230.40	197,842	0.04	2.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.75	10n	37,923.48	513,825	0.46	6.39	0.00	0.00	0.00	0.00	0.08	8.08	0.00	0.00
		30n	0.00	0	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	0.00	0	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.75	0.25	10n	906,273.05	32,881,824*	11.56	388.05	0.22	13.88	10.23	1000.00	0.18	8.47	0.05	4.55
		30n	233,616.08	7,809,830	2.90	104.58	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	187,185.72	11,482,136	2.34	137.83	0.00	0.00	0.00	0.00	0.50	50.00	0.00	0.00
	0.50	10n	52,821.20	696,483	0.68	9.20	0.37	35.22	2.61	200.00	0.01	0.68	0.01	0.68
		30n	1254.95	94,666	0.02	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	43.32	3826	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.75	10n	11,824.61	282,636	0.15	3.31	0.61	56.25	0.05	5.19	0.14	8.33	0.05	5.19
		30n	70.48	5561	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		50n	0.00	0	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

An asterisk means an instance with number of nodes over 100,000,000.

Table 4
The performance of the genetic algorithms with $n = 50$, $\lambda = 0.2$, and $\tau = 0.5$.

R	P	M	GA_1				GA_2				GA_3			
			RDP		CPU time		RDP		CPU time		RDP		CPU time	
			Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
0.25	0.25	10n	1.05	10.60	11.53	11.92	1.55	14.73	11.65	12.00	0.67	10.44	11.52	11.91
		30n	0.07	5.87	5.53	12.06	0.00	0.00	5.42	12.11	0.15	8.74	5.56	12.25
		50n	0.22	11.74	6.79	11.98	0.13	11.74	6.66	12.17	0.09	5.18	6.81	12.13
	0.50	10n	2.66	38.94	11.39	11.80	2.73	51.34	11.50	12.02	1.65	51.34	11.33	11.81
		30n	0.00	0.00	0.16	0.25	0.00	0.00	0.00	0.02	0.00	0.00	0.15	0.25
		50n	0.00	0.00	0.15	0.23	0.00	0.00	0.00	0.00	0.00	0.00	0.15	0.27
	0.75	10n	1.99	56.07	10.88	11.83	1.57	57.10	10.90	11.81	1.21	28.87	10.72	11.69
		30n	0.00	0.00	0.05	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.07	0.16
		50n	0.00	0.00	0.04	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.11
0.50	0.25	10n	3.35	79.15	10.95	12.06	3.48	110.54	11.05	12.22	0.50	12.44	10.89	12.06
		30n	0.00	0.00	0.88	12.05	0.00	0.00	0.63	12.06	0.00	0.00	0.76	12.05
		50n	0.00	0.00	1.18	11.98	0.00	0.00	0.85	11.98	0.00	0.00	1.08	11.84
	0.50	10n	2.17	108.71	6.29	11.69	2.46	96.83	6.27	11.89	0.90	46.24	6.00	11.61
		30n	0.00	0.00	0.18	0.34	0.00	0.00	0.00	0.03	0.00	0.00	0.12	0.27
		50n	0.00	0.00	0.15	0.25	0.00	0.00	0.00	0.02	0.00	0.00	0.11	0.22
	0.75	10n	0.52	27.97	2.58	11.53	0.00	0.00	2.29	11.61	1.01	54.09	2.12	11.41
		30n	0.00	0.00	0.06	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.13
		50n	0.00	0.00	0.05	0.09	0.00	0.00	0.00	0.02	0.00	0.00	0.06	0.11
0.75	0.25	10n	1.61	127.27	4.53	12.16	0.78	47.91	4.53	12.53	0.26	22.71	4.24	12.16
		30n	0.00	0.00	0.53	12.00	0.00	0.00	0.34	11.91	0.00	0.00	0.27	11.89
		50n	0.04	4.00	0.49	12.13	0.00	0.00	0.14	12.00	0.04	4.00	0.26	12.05
	0.50	10n	0.08	7.52	1.31	11.44	0.03	2.62	1.21	11.63	0.00	0.00	0.80	11.41
		30n	0.00	0.00	0.23	0.48	0.00	0.00	0.01	0.27	0.00	0.00	0.07	0.22
		50n	0.00	0.00	0.16	0.31	0.00	0.00	0.00	0.03	0.00	0.00	0.06	0.14
	0.75	10n	0.00	0.00	0.49	1.38	0.00	0.00	0.26	4.13	0.00	0.00	0.06	0.36
		30n	0.00	0.00	0.09	0.25	0.00	0.00	0.01	0.11	0.00	0.00	0.04	0.08
		50n	0.00	0.00	0.06	0.11	0.00	0.00	0.00	0.02	0.00	0.00	0.04	0.06

recommend to use the best sequence from three GAs, $GA^* = \min\{GA_1, GA_2, GA_3\}$, as the approximate solution, since they are all finished within a second and its mean error percentages are less than 0.2% for all the tested cases.

The last part of the computational experiments is to test the performance of GAs when the number of jobs is large. The number of jobs is set at 50 and 100. A set of 100 instances is tested, and the results are presented in Tables 4 and 5. The mean and the

Table 5
The performance of the genetic algorithms with $n = 100$, $\lambda = 0.2$, and $\tau = 0.5$.

R	P	M	GA_1				GA_2				GA_3			
			RDP		CPU time		RDP		CPU time		RDP		CPU time	
			Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
0.25	0.25	10n	2.72	14.55	73.68	76.53	4.13	44.99	73.76	75.97	1.24	9.41	73.69	76.33
		30n	0.19	15.79	24.28	75.70	0.25	12.31	22.96	75.31	0.22	15.79	23.69	75.92
		50n	3.89	216.67	20.89	76.47	0.00	0.00	18.08	74.33	0.00	0.00	20.46	75.95
	0.50	10n	3.44	40.27	72.93	75.45	2.80	40.27	72.50	75.55	1.77	31.56	72.11	75.17
		30n	0.00	0.00	1.26	1.73	0.00	0.00	0.00	0.05	0.00	0.00	1.18	1.84
		50n	0.00	0.00	1.13	1.55	0.00	0.00	0.00	0.02	0.00	0.00	1.05	1.59
	0.75	10n	2.59	36.81	70.56	75.08	1.61	48.99	69.80	74.70	1.95	36.81	69.63	74.55
		30n	0.00	0.00	0.56	1.23	0.00	0.00	0.00	0.02	0.00	0.00	0.53	0.95
		50n	0.00	0.00	0.45	0.69	0.00	0.00	0.00	0.02	0.00	0.00	0.48	0.77
0.50	0.25	10n	13.64	813.85	71.68	76.97	13.00	726.15	71.54	76.33	5.14	97.06	71.29	76.48
		30n	0.00	0.00	3.08	4.66	0.00	0.00	1.59	4.39	0.00	0.00	1.88	3.48
		50n	0.00	0.00	2.76	5.02	0.00	0.00	0.08	0.23	0.00	0.00	1.75	3.44
	0.50	10n	2.52	90.38	26.04	74.42	3.72	90.38	23.81	73.97	7.68	432.82	22.28	73.95
		30n	0.00	0.00	1.53	2.22	0.00	0.00	0.00	0.00	0.00	0.00	0.87	1.52
		50n	0.00	0.00	1.19	1.63	0.00	0.00	0.00	0.02	0.00	0.00	0.74	1.19
	0.75	10n	0.02	1.54	5.26	70.39	0.00	0.00	2.51	69.06	0.00	0.00	1.34	70.03
		30n	0.00	0.00	0.80	1.50	0.00	0.00	0.00	0.00	0.00	0.00	0.39	0.92
		50n	0.00	0.00	0.48	0.78	0.00	0.00	0.00	0.02	0.00	0.00	0.36	0.77
0.75	0.25	10n	2.13	124.38	21.55	77.30	1.90	107.72	21.00	75.75	0.19	7.38	17.48	76.13
		30n	0.00	0.00	3.07	5.25	0.00	0.00	2.31	5.98	0.00	0.00	0.82	2.00
		50n	0.00	0.00	2.68	7.19	0.00	0.00	0.06	0.23	0.00	0.00	0.74	3.72
	0.50	10n	0.00	0.00	4.77	17.44	0.00	0.00	2.45	7.20	0.00	0.00	0.58	2.78
		30n	0.00	0.00	1.95	2.89	0.00	0.00	0.05	2.86	0.00	0.00	0.43	1.48
		50n	0.00	0.00	1.28	2.02	0.00	0.00	0.01	0.05	0.00	0.00	0.36	0.78
	0.75	10n	0.00	0.00	3.41	5.69	0.00	0.00	0.98	4.66	0.00	0.00	0.22	0.97
		30n	0.00	0.00	1.13	2.06	0.00	0.00	0.51	27.25	0.00	0.00	0.18	0.50
		50n	0.00	0.00	0.60	1.25	0.00	0.00	0.01	0.08	0.00	0.00	0.18	0.39

maximum relative deviation percentages, and the mean and the maximum CPU time (in second) are recorded. The relative deviation percentage (RDP) of the solution produced by GA_i is calculated as

$$\frac{(V_i - \min\{V_1, V_2, V_3\})}{\min\{V_1, V_2, V_3\}} \times 100\%$$

for $i = 1, 2, 3$ where V_i is the objective value of the sequence from GA_i. It is observed that the execution times of the *GA*s are about the same. It is seen that GA_3 has the best overall performance and the trend becomes more significant when the number of jobs increases. The execution times of the *GA*s are about the same. It takes about 12 s for an instance of 50 jobs and 75 s for an instance of 100 jobs.

5. Conclusion

In this paper, we studied a two-agent scheduling problem on a single machine with release time. The objective is to minimize the total tardiness of jobs from the first agent given that the maximum tardiness of jobs from the second agent cannot exceed a given upper bound. Computational results show that the branch-and-bound algorithm could solve most of the problems with 16 jobs within a reasonable amount of time. In addition, it shows that the performance of the combined genetic algorithm is very good with mean error percentages of less than 0.2% for all the cases. Considering objective functions, other than the two examined in this paper or extending the single-machine case to other machine environments would be an interesting topic for future research.

Acknowledgements

The authors are grateful to the editor and the referees, whose constructive comments have led to a substantial improvement in the presentation of the paper. This work was supported by the NSC of Taiwan, ROC, under NSC 100-2221-E-035-029-MY3.

References

[1] K.R. Baker, J.C. Smith, A multiple-criterion model for machine scheduling, Journal of Scheduling 6 (2003) 7–16.
[2] M.A. Kubzin, V.A. Strusevich, Planning machine maintenance in two-machine shop scheduling, Operations Research 54 (2006) 789–800.
[3] C.R. Meiners, E. Torng, Mixed criteria packet scheduling, Proceedings of the Third International Conference on Algorithmic Aspects in Information and Management, Lecture Notes in Computer Science 4508 (2009) 120–133.
[4] M.J. Soomer, G.J. Franx, Scheduling aircraft landings using airlines' preferences, European Journal of Operations Research 190 (2008) 277–291.
[5] J.Y.T. Leung, M. Pinedo, G.H. Wan, Competitive two agents scheduling and its applications, Operations Research 58 (2010) 458–469.
[6] A. Agnetis, P.B. Mirchandani, D. Pacciarelli, A. Pacifici, Scheduling problems with two competing agents, Operations Research 52 (2004) 229–242.
[7] J.J. Yuan, W.P. Shang, Q. Feng, A note on the scheduling with two families of jobs, Journal of Scheduling 8 (2005) 537–542.
[8] C.T. Ng, T.C.E. Cheng, J.J. Yuan, A note on the complexity of the problem of two-agent scheduling on a single machine, Journal of Combinatorial Optimization 12 (2006) 387–394.
[9] T.C.E. Cheng, C.T. Ng, J.J. Yuan, Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs, Theoretical Computer Science 362 (2006) 273–281.
[10] A. Agnetis, D. Pacciarelli, A. Pacifici, Multi-agent single machine scheduling, Annals of Operations Research 150 (2007) 3–15.
[11] T.C.E. Cheng, C.T. Ng, J.J. Yuan, Multi-agent scheduling on a single machine with max-form criteria, European Journal of Operational Research 188 (2008) 603–609.
[12] P. Liu, L. Tang, Two-agent scheduling with linear deteriorating jobs on a single machine, Lecture Notes in Computer Science 5092 (2008) 642–650.
[13] A. Agnetis, G. Pascale, D. Pacciarelli, A Lagrangian approach to single-machine scheduling problems with two competing agents, Journal of Scheduling 12 (2009) 401–415.
[14] K.B. Lee, B.C. Choi, J.Y.T. Leung, M.L. Pinedo, Approximation algorithms for multi-agent scheduling to minimize total weighted completion time, Information Processing Letters 109 (2009) 913–917.
[15] P. Liu, L. Tang, X. Zhou, Two-agent group scheduling with deteriorating jobs on a single machine, International Journal of Advanced Manufacturing Technology 47 (2010) 657–664.
[16] W.C. Lee, W.J. Wang, Y.R. Shiau, C.C. Wu, A single-machine scheduling problem with two-agent and deteriorating jobs, Applied Mathematical Modelling 34 (2010) 3098–3107.
[17] W.C. Lee, S.K. Chen, C.C. Wu, Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem, Expert Systems with Applications 37 (2010) 6594–6601.
[18] P. Liu, X. Zhou, L. Tang, Two-agent single-machine scheduling with position-dependent processing times, International Journal of Advanced Manufacturing Technology 48 (2010) 325–331.
[19] G. Wan, S.R. Vakati, J.Y.T. Leung, M. Pinedo, Scheduling two agents with controllable processing times, European Journal of Operational Research 205 (2010) 528–539.
[20] B. Mor, G. Mosheiov, Scheduling problems with two competing agents to minimize minmax and minsum earliness measures, European Journal of Operational Research 206 (2010) 540–546.
[21] W.C. Lee, S.K. Chen, C.W. Chen, C.C. Wu, A two-machine flowshop problem with two agents, Computers and Operations Research 38 (2011) 98–104.
[22] P. Liu, N. Yi, X. Zhou, Two-agent single-machine scheduling problems under increasing linear deterioration, Applied Mathematical Modelling 35 (2011) 2290–2296.
[23] Q.Q. Nong, T.C.E. Cheng, C.T. Ng, Two agent scheduling to minimize the total cost, European Journal of Operational Research 215 (2011) 39–44.
[24] Y.Q. Yin, S.R. Cheng, C.C. Wu, Scheduling problems with two agents and a linear non-increasing deterioration to minimize earliness penalties, Information Sciences (2011), http://dx.doi.org/10.1016/j.ins.2011.11.035.
[25] B. Mor, G. Mosheiov, Single machine batch scheduling with two competing agents to minimize total flowtime, European Journal of Operational Research 215 (2011) 524–531.
[26] C.C. Wu, S.K. Huang, W.C. Lee, Two-agent scheduling with learning consideration, Computers & Industrial Engineering 61 (2011) 1324–1335.
[27] T.C.E. Cheng, S.R. Cheng, W.H. Wu, P.H. Hsu, C.C. Wu, A two-agent single-machine scheduling problem with truncated sum-of-processing-times-based learning considerations, Computers & Industrial Engineering 60 (2011) 534–541.
[28] D.C. Li, P.H. Hsu, Solving a two-agent single-machine scheduling problem considering learning effect, Computers & Operations Research 39 (2012) 1644–1651.
[29] J. Du, J.Y.T. Leung, Minimizing total tardiness on one machine is NP-hard, Mathematics of Operations Research 15 (1990) 483–495.
[30] K. Li, Y. Shi, S.L. Yang, B.Y. Cheng, Parallel machine scheduling problem to minimize the makespan with resource dependent processing times, Applied Soft Computing 11 (2011) 5551–5557.
[31] B. Yua, Z. Yang, X. Sun, B. Yao, Q. Zeng, E. Jeppesen, Parallel genetic algorithm in bus route headway optimization, Applied Soft Computing 11 (2011) 5081–5091.
[32] R. Yusof, M. Khalid, G.T. Hui, S.M. Yusof, M.F. Othman, Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm, Applied Soft Computing 11 (2011) 5782–5792.
[33] T.J. Hsieh, H.F. Hsiao, W.C. Yeh, Forecasting stock markets using wavelet transforms and recurrent neural networks: an integrated system based on artificial bee colony algorithm, Applied Soft Computing 11 (2011) 2510–2525.
[34] C.Y. Low, C.J. Hsu, C.T. Su, A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance, Expert Systems with Applications 37 (2010) 6429–6434.
[35] J. Behnamian, S.M.T.F. Ghomi, F. Jolai, O. Amirtaheri, Minimizing makespan on a three-machine flowshop batch scheduling problem with transportation using genetic algorithm, Applied Soft Computing 11 (2012) 768–777.
[36] P.C. Chang, S.H. Chen, Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times, Applied Soft Computing 11 (2011) 1263–1274.
[37] D. Lei, Simplified multi-objective genetic algorithms for stochastic job shop scheduling, Applied Soft Computing 11 (2011) 4991–4996.
[38] O. Engin, G. Ceran, M.K. Yilmaz, An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems, Applied Soft Computing 11 (2011) 3056–3065.
[39] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA Journal of Computing 6 (1994) 154–160.
[40] A. Homaifar, C. Qi, S. Lai, Constrained optimization via genetic algorithms, Simulation 36 (1994) 242–254.
[41] C.B. Chu, A branch-and-bound algorithm to minimize total flow time with unequal release dates, Naval Research Logistics 39 (1992) 859–875.