RAIRO-Oper. Res. 54 (2020) 307–323
https://doi.org/10.1051/ro/2019088

RAIRO Operations Research
www.rairo-ro.org

A THREE-AGENT SCHEDULING PROBLEM FOR MINIMIZING THE FLOW TIME ON TWO MACHINES

Wen-Chiung Lee[1] and Jen-Ya Wang[2,*]

Abstract. This study introduces a two-machine three-agent scheduling problem. We aim to minimize the total tardiness of jobs from agent 1 subject to that the maximum completion time of jobs from agent 2 cannot exceed a given limit and that two maintenance activities from agent 3 must be conducted within two maintenance windows. Due to the NP-hardness of this problem, a genetic algorithm (named GA^+) is proposed to obtain approximate solutions. On the other hand, a branch-and-bound algorithm (named B&B) is developed to generate the optimal solutions. When the problem size is small, we use B&B to verify the solution quality of GA^+. When the number of jobs is large, a relative deviation is proposed to show the gap between GA^+ and another ordinary genetic algorithm. Experimental results show that the proposed genetic algorithm can generate approximate solutions by consuming reasonable execution time.

Mathematics Subject Classification. 90C05, 90-08.

Received December 30, 2018. Accepted September 6, 2019.

1. Introduction

Multi-agent scheduling is of significant importance in the real word. As first indicated by Baker *et al.* [3], an agent (*e.g.*, R&D department) might necessarily complete its jobs before a given date, while another agent (*e.g.*, sales department) could not afford any indemnification caused by any uncontrolled tardiness. Consequently, these agents needed to learn how to share limited resources (*e.g.*, machines or workers). It is clear that job scheduling is an effective way to resolve the above issue. Multi-agent scheduling is therefore called for and deserves to be studied in greater detail.

At the beginning of multi-agent scheduling, only two competing agents were considered. Even so, most of such problems are NP-hard (*e.g.*, [5,49,56]). Consequently, the issues of two-agent scheduling have been widely and thoroughly discussed for decades. However, in the real world, the number of objectives and constraints are far more than two. That is, two-agent models are too simple to represent the real world. If there are more agents, we can make multi-agent scheduling more practical. For example, research [22,40,45] recently started to discuss the issues of three-agent scheduling. Consequently, they considered three agents and made their models

Keywords. Multi-agent scheduling, maintenance scheduling, branch-and-bound algorithm, genetic algorithm, solution quality.

[1] Department of Statistics, Feng Chia University, 40724 Taichung, Taiwan, R.O.C.

[2] Department of Computer Science and Information Management, Hungkuang University, 43302 Taichung, Taiwan, R.O.C.

*Corresponding author: `jywang@sunrise.hk.edu.tw`

more realistic but more complicated. For more recent information about multi-agent scheduling, we can refer to [14, 26, 34, 39, 42, 48, 50].

Total tardiness is usually an important consideration for multi-agent scheduling. For example, Ahmadizar *et al.* [2] proposed a two-agent scheduling problem in which the objective of agent 1 was to minimize his/her earliness and tardiness at the same time. On the other hand, the other agent must keep his/her maximum tardiness and earliness under a given limit. If a job from agent 1 was completed too early or too late, a penalty would be inflicted upon the agent. Yang *et al.* [52] addressed a multi-agent scheduling problem in a cloud computing environment. Agents competed for limited resources (*e.g.*, processors) and intended to minimize the total tardiness. They formulated this problem as an integer programming problem and developed a tabu search method and a genetic algorithm to solve the problem near optimally. Wang *et al.* [47] also considered two-agent scheduling on a single machine. They proposed nine different combinations of objectives, one being total tardiness. In this study, the due date assignment technique was employed to achieve a lower total tardiness for agent 1. In light of the above observation, we learn that total tardiness is an important indicator for evaluating a production system. For more information about tardiness and competing agents, we can refer to related articles; *e.g.*, [8, 11, 23–25, 28, 39, 46, 54].

At the same time, a maximum completion time might be requested by another agent. A maximum completion time (or makespan) is a useful indicator to measure a scheduling model; *e.g.*, inventory, productivity, or resource deployment. Therefore, many exact or approximation algorithms are proposed to reduce the indicator for an agent. For example, Torkashvand *et al.* [44] introduced a flowshop scheduling problem with two competing agents. The objective of agent 1 was to minimize the makespan. Since such a problem was time-consuming, they proposed a metaheuristic algorithm to generate approximate solutions. On the other hand, in [30], the authors considered the makespan as a constraint of an agent in a flowshop scheduling problem. Unless the constraint was satisfied, then the other agent would not minimize his/her objective. For more research on two-agent makespan minimization, we can refer to [1, 9, 27, 33, 35].

On the other hand, an agent may shut down machines for maintenance during designated periods. In general, maintenance activities are of two types, continuous (minor) maintenance and overhaul (major) maintenance, and they should be inserted into ordinary jobs. In [42], minor cleaning activities made a single machine unavailable for specified periods. One agent aimed to minimize the total deviation of completion times subject to these periods of unavailability to perform the other agent's cleaning activities. Due to the NP-hardness of this problem, they employed a metaheuristic algorithm to solve this problem. Yu *et al.* [58] considered that major maintenance is needed once a fixed number of jobs is completed. Since the behaviors of maintenance were modeled in a simpler form, the problem could be solved in polynomial time. In [4], both major preventive maintenance (time-based) and minor anomaly detection (condition-based) were considered. Because of the two kinds of maintenance activities, this problem could not be solved with traditional approaches. That is, time-based and condition-based maintenance needed to be treated as a whole. For more recent information about maintenance scheduling, please refer to [13, 16, 31, 37, 41, 43, 51, 53, 60].

Total tardiness minimization is also an important topic for two-machine flowshop scheduling. Kim [19] developed some properties and a lower bound and proposed a branch-and-bound algorithm to minimize the total tardiness on two flowshop machines. Pan *et al.* [36] considered different constraints and assumptions and also employed the same technique to minimize the total tardiness on two flowshop machines. Schaller [38] aimed to optimally solve a total tardiness minimization problem with different constraints on two flowshop machines. Again, they proposed three useful dominance rules and a lower bound to enhance their branch-and-bound algorithm. Since the constraints of the above problems are all different, they needed to tailor their own algorithms to individual needs. For more recent studies about total tardiness minimization on two machines, readers can refer to [10, 15, 17, 20, 29].

Among the above issues, only two are considered at a time. That is, a few studies, such as [21, 22, 40, 45], considered three of them as a whole. For those three-agent scheduling problems, they considered a single machine only. Therefore, we examine the idea of multiple competing agents in a two-machine flowshop environment. Since total tardiness is directly relevant to customer satisfaction and business income, we set it as an objective. On the other hand, in most situations, some orders may have imposed deadlines, so we set maximum completion time (*i.e.*, makespan) as a constraint. Last of all, most machines require preventive maintenance, so two fixed maintenance windows are considered as another constraint.

In this study, a three-agent flow-shop problem is introduced. This is because two agents cannot fulfill the needs of the real world sufficiently. For example, research [6, 7] considered maximum tardiness and periodic maintenance in a textile company. Without the flexibility of more agents, total completion time could not be included in this model. Consequently, we choose a three-agent model to meet the requirements of the real world. In our model, agent 1 aims to minimize the total tardiness of his/her own jobs subject to the constraints that the maximum completion time of jobs from agent 2 is below a given limit and that the two machines must be maintained within two specified periods. Due to the NP-hardness of the problem, we propose a genetic algorithm to generate approximate solutions. Moreover, we develop a relative error bound to ensure the solution quality of the proposed genetic algorithm. Compared with other metaheuristic algorithms (*e.g.*, PSO), GA has lower tendency for premature convergence and requires less density of search space [18]. Therefore, we develop a genetic algorithm to generate approximate solutions. On the other hand, compared with other exact algorithms, a branch-and-branch algorithm is relatively time-efficient if effective dominance rules and lower bounds are proposed. Hence we develop a branch-and-branch algorithm to generate the optimal solutions for generating the optimal solutions when the problem size is small.

2. Problem formulation

The three-agent two-machine flowshop problem is formulated as follows. There are n non-preemptive jobs from three agents to be processed by two machines. For each job j, machine 1 first takes processing time p_j^1 and then machine 2 takes processing time p_j^2 to process it respectively. Each machine processes only one job at a time, and a job can be processed by each machine at most once. In addition, the position order of all the jobs processed by machine 1 is the same as that by machine 2. Agent ag_1 aims to minimize the total tardiness of his/her n_1 jobs and each is tagged with an individual due date d_j for $j = 1, 2, \ldots, n_1$. Agent ag_2 must complete all his/her n_2 jobs (numbered $n_1+1, n_1+2, \ldots, n_1+n_2$) before a limit M. Moreover, there are two maintenance jobs from ag_3; *i.e.*, job $n-1$ needs to be conducted within a maintenance window $[a_{n-1}^1, b_{n-1}^1]$ on machine 1 and job n needs to be conducted within $[a_n^2, b_n^2]$ on machine 2. For simplicity, let AG_i denote the set of jobs from agent ag_i for $i = 1, 2, 3$. Note that we have $n = n_1 + n_2 + 2$, $p_{n-1}^2 = 0$, and $p_n^1 = 0$. For job j in schedule π, let $C_j(\pi)$ denote its completion time and $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ denote the tardiness if $j \in \mathrm{AG}_1$. For simplicity, we use $C_j^1(\pi)$ to denote the time that machine 1 finishes job j. For each job $j \in \mathrm{AG}_1 \cup \mathrm{AG}_2$, its starting time on machine 2 cannot be earlier than $C_j^1(\pi)$. Under the above assumptions and constraints, the problem can be presented as

$$\min\ f(\pi) = \sum_{j \in \mathrm{AG}_1} T_j(\pi) \tag{2.1}$$

$$\text{s.t.} \tag{2.2}$$

$$\max\nolimits_{j \in \mathrm{AG}_2}\{C_j(\pi)\} \le M, \tag{2.3}$$

$$C_{n-1}^1(\pi) \in [a_{n-1}^1 + p_{n-1}^1, b_{n-1}^1], \tag{2.4}$$

$$C_n(\pi) \in [a_n^2 + p_n^2, b_n^2]. \tag{2.5}$$

Figure 1 shows a problem instance. There are seven jobs, and $\mathrm{AG}_1 = \{1, 2, 3, 4\}$, $\mathrm{AG}_2 = \{5\}$, and $\mathrm{AG}_3 = \{6, 7\}$, (*i.e.*, $n_1 = 4$, $n_2 = 1$, $n_3 = 2$) and $n = 7$. Let $p_j^1 = 2, 1, 4, 5, 4, 2, 0, p_j^2 = 4, 1, 1, 1, 1, 0, 2$, for $j = 1, 2, \ldots, 7$, and $d_j = 5,\ 5,\ 10,\ 15$, for $j \in \mathrm{AG}_1$. Moreover, let $M = 8$, $a_6^1 = 8$, $b_6^1 = 10$, $a_7^2 = 9$, and $b_7^2 = 13$. Then, for the schedule $\pi = (1, 2, 5, 6, 7, 3, 4)$, the total tardiness is 13 $(= 1 + 2 + 5 + 5)$. Note that the starting time of maintenance activity 7 is not affected by maintenance activity 6. On the other hand, there might be idle intervals due to the influences of maintenance window and flowshop.

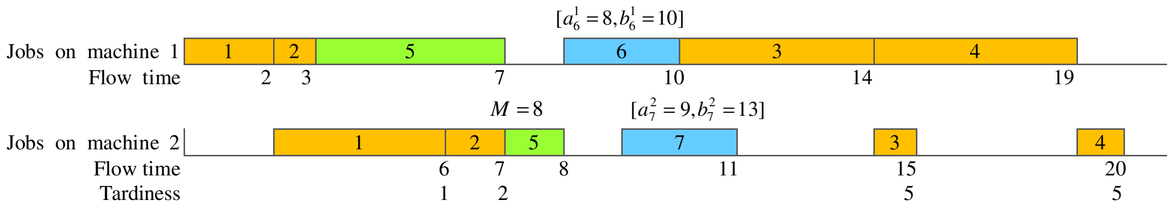

FIGURE 1. A problem instance.

In general, such a scheduling problem with maintenance is NP-hard [55, 57, 59]. Even if only one single machine needs to be maintained within a maintenance window, the problem is still NP-hard. Consequently, some metaheuristic algorithms are required when the problem size is large.

3. Branch-and-bound algorithm

In this section, we develop an exact optimization algorithm to obtain the optimal schedules. First, several dominance rules are derived. Then, a lower bound algorithm is proposed. Finally, a branch-and-bound algorithm named B&B is proposed.

To derive dominance rules, we first let $\pi = (\alpha, j, i, \beta)$ and $\pi' = (\alpha, i, j, \beta)$ be two schedules, where i and j are two adjacent jobs, α is a determined partial sequence, and β is an undetermined partial sequence. For convenience, let n_β be the number of jobs in β and t_α^1 (t_α^2) be the completion time of the last job in α on machine 1 (machine 2). To show that π' dominates π, we need to ensure that either $C_j(\pi') \leq C_i(\pi)$ and $T_i(\pi') + T_j(\pi') < T_i(\pi) + T_j(\pi)$ hold or $C_j(\pi') < C_i(\pi)$ and $T_i(\pi') + T_j(\pi') \leq T_i(\pi) + T_j(\pi)$ hold. The related dominance rules are organized into 14 cases according to their completion times and tardinesses. Since their proofs are similar, only the proof of the second dominance rule is given below.

Case 1. We consider that both jobs i and j are from ag_1 and processed in time; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') = 0$, $T_i(\pi) = 0$, $T_j(\pi') = 0$, and $T_j(\pi) = 0$. If job j in π' is always completed earlier than i in π and both have zero tardiness, we let job i proceed job j.

Rule 1. If $C_j(\pi') < C_i(\pi)$, then π' dominates π.

Case 2. We consider that both jobs are from ag_1 and job j is tardy in both schedules; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') = 0$, $T_i(\pi) = 0$, $T_j(\pi') > 0$, and $T_j(\pi) > 0$. If job j in π' is always completed earlier than i in π and will not cause any extra tardiness, we let job i proceed job j.

Rule 2. If $C_j(\pi') < C_i(\pi)$ and $T_j(\pi') \leq T_j(\pi)$, then π' dominates π.

Proof. Since $C_j(\pi') < C_i(\pi)$, the starting time of the remaining jobs (*i.e.*, β) of π' is earlier than that of π. That is, $\sum_{k\in\beta} T_k(\pi') \leq \sum_{k\in\beta} T_k(\pi)$ for both optimal sequences of β in π and π'. In addition, we have $T_i(\pi') + T_j(\pi') \leq T_j(\pi) + T_i(\pi)$ and $\sum_{k\in\alpha} T_k(\pi') = \sum_{k\in\alpha} T_k(\pi)$. Therefore, $\sum_k T_k(\pi') \leq \sum_k T_k(\pi)$. The proof is complete. □

Rule 3. If $C_j(\pi') \leq C_i(\pi)$ and $T_j(\pi') < T_j(\pi)$, then π' dominates π.

Case 3. We consider that both jobs are from ag_1 and only job i is tardy in schedule π; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') = 0$, $T_i(\pi) > 0$, $T_j(\pi') = 0$, and $T_j(\pi) = 0$. If the tardiness of job i in π can be improved in π' and job j is always completed in time, we let job i proceed job j.

Rule 4. If $C_j(\pi') \leq C_i(\pi)$, then π' dominates π.

Case 4. We consider that both jobs are from ag_1, job i is tardy in schedule π, and job j is tardy in schedule π'; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') = 0$, $T_i(\pi) > 0$, $T_j(\pi') > 0$, and $T_j(\pi) = 0$.

Rule 5. If $C_j(\pi') \leq C_i(\pi)$ and $T_j(\pi') < T_i(\pi)$, then π' dominates π.

Rule 6. If $C_j(\pi') < C_i(\pi)$ and $T_j(\pi') \leq T_i(\pi)$, then π' dominates π.

Case 5. We consider that both jobs are from ag_1, job i is tardy in schedule π, and job j is always tardy in both schedules; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') = 0$, $T_i(\pi) > 0$, $T_j(\pi') > 0$, and $T_j(\pi) > 0$. If job j in π' can be completed early and make no harm to the total tardiness, we let job i proceed job j.

Rule 7. If $C_j(\pi') \leq C_i(\pi)$ and $T_j(\pi') < T_i(\pi) + T_j(\pi)$, then π' dominates π.

Rule 8 . If $C_j(\pi') < C_i(\pi)$ and $T_j(\pi') \leq T_i(\pi) + T_j(\pi)$, then π' dominates π.

Case 6. We consider that both jobs are from ag_1 and job i is always tardy in both schedules; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') > 0$, $T_i(\pi) > 0$, $T_j(\pi') = 0$, and $T_j(\pi) = 0$. If completion time is improved and tardiness remains unchanged or *vice versa*, we let job i proceed job j.

Rule 9. If $C_j(\pi') \leq C_i(\pi)$ and $T_i(\pi') < T_i(\pi)$, then π' dominates π.

Rule 10. If $C_j(\pi') < C_i(\pi)$ and $T_i(\pi') \leq T_i(\pi)$, then π' dominates π.

Case 7. We consider that both jobs are from ag_1, job i is always tardy, and job j is always tardy in schedule π'; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') > 0$, $T_i(\pi) > 0$, $T_j(\pi') > 0$, and $T_j(\pi) = 0$. If the interchange leads to either a lower tardiness or completion time, we let job i proceed job j.

Rule 11. If $C_j(\pi') \leq C_i(\pi)$ and $T_i(\pi') + T_j(\pi') < T_i(\pi)$, then π' dominates π.

Rule 12. If $C_j(\pi') < C_i(\pi)$ and $T_i(\pi') + T_j(\pi') \leq T_i(\pi)$, then π' dominates π.

Case 8. We consider that both jobs are from ag_1 and they are always tardy; *i.e.*, $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $T_i(\pi') > 0$, $T_i(\pi) > 0$, $T_j(\pi') > 0$, and $T_j(\pi) > 0$. In this case, the rules are the same as those in Case 1. Consequently, we omit them here.

Case 9. We consider that both jobs are from ag_2; *i.e.*, $i \in \mathrm{AG}_2$ and $j \in \mathrm{AG}_2$. If job j in π'can be completed earlier than i in π or job j has a larger Id, we let job i proceed job j.

Rule 13. If $C_j(\pi') < C_i(\pi)$ and $C_j(\pi') \leq M$, then π' dominates π.

Rule 14. If $C_j(\pi') = C_i(\pi)$, $C_j(\pi') \leq M$, and $i < j$, then π' dominates π.

Case 10. We consider that jobs i and j are from different agents; *i.e.*, $i \in \mathrm{AG}_1$ and $j \in \mathrm{AG}_2$. In this case, we let a job from agent 2 be processed as late as possible.

Rule 15. If $C_j(\pi') \leq C_i(\pi)$ and $C_j(\pi') \leq M$, then π' dominates π.

Case 11. We consider that jobs i and j are from different agents; *i.e.*, $i \in \mathrm{AG}_2$ and $j \in \mathrm{AG}_1$. If the completion time of a job from agent 2 exceeds the limit, this job can be advanced.

Rule 16. If $C_j(\pi') \leq C_i(\pi)$, $C_i(\pi) > M$, and $C_i(\pi') \leq M$, then π' dominates π.

Case 12. We consider that job i is a maintenance job; *i.e.*, $i \in \mathrm{AG}_3$ and $j \notin \mathrm{AG}_3$. In this case, if a maintenance activity i cannot be finished in time, we let maintenance activity proceed an ordinary job.

Rule 17. If $i = n-1$ and $j \in \mathrm{AG}_1$ are processed by machine 1, $C_j(\pi') \leq C_i(\pi)$, $C_i^1(\pi) > b_{n-1}^1$, and $C_i^1(\pi') \leq b_{n-1}^1$, then π' dominates π.

Rule 18. If $i = n$ and $j \in \mathrm{AG}_1$ are processed by machine 2, $C_j(\pi') \leq C_i(\pi)$, $C_i(\pi) > b_n^2$, and $C_i(\pi') \leq b_n^2$, then π' dominates π.

Rule 19. If $i = n-1$ and $j \in \mathrm{AG}_2$ are processed by machine 1, $C_j(\pi') \leq C_i(\pi)$, $C_i^1(\pi) > b_{n-1}^1$, $C_i^1(\pi') \leq b_{n-1}^1$, and $C_j(\pi') \leq M$, then π' dominates π.

Rule 20. If $i = n$ and $j \in \mathrm{AG}_2$ are processed by machine 2, $C_j(\pi') \leq C_i(\pi)$, $C_i(\pi) > b_n^2$, $C_i(\pi') \leq b_n^2$, and $C_j(\pi') \leq M$, then π' dominates π.

Case 13. We consider that job j is a maintenance job; *i.e.*, $i \notin \mathrm{AG}_3$ and $j \in \mathrm{AG}_3$. In this case, we perform a maintenance activity as late as possible.

Rule 21. If $j = n-1$ and $i \in \mathrm{AG}_1$ are processed by machine 1, $C_j(\pi') \leq C_i(\pi)$, and $C_j^1(\pi') \leq b_{n-1}^1$, then π' dominates π.

Rule 22. If $j = n$ and $i \in \mathrm{AG}_1$ are processed by machine 2, $C_j(\pi') \leq C_i(\pi)$, and $C_j(\pi') \leq b_n^2$, then π' dominates π.

Rule 23. If $j = n-1$ and $i \in \mathrm{AG}_2$ are processed by machine 1, $C_j(\pi') \leq C_i(\pi)$, $C_i(\pi) > M$, $C_j^1(\pi') \leq b_{n-1}^1$, and $C_i(\pi') \leq M$, then π' dominates π.

Rule 24. If $j = n$ and $i \in \mathrm{AG}_2$ are processed by machine 2, $C_j(\pi') \leq C_i(\pi)$, $C_i(\pi) > M$, $C_j(\pi') \leq b_n$, and $C_i(\pi') \leq M$, then π' dominates π.

Rule 25. If $j = n-1$ and $i \in \mathrm{AG}_2$ are processed by machine 1, $C_j(\pi') \leq C_i(\pi)$, $t^1_\alpha(\pi) < a^1_{n-1}$, $C^1_j(\pi') \leq b^1_{n-1}$, and $C_i(\pi') \leq M$, then π' dominates π.
Rule 26. If $j = n$ and $i \in \mathrm{AG}_2$ are processed by machine 2, $C_j(\pi') \leq C_i(\pi)$, $t^2_\alpha(\pi) < a^2_n$, $C^1_j(\pi') \leq b^1_{n-1}$, and $C_i(\pi') \leq M$, then π' dominates π.

Case 14. There are some other situations which do not belong to the above cases. We list these properties as follows.
Rule 27. If $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $p^1_i = p^1_j$, $p^2_i = p^2_j$, $d_i = d_j$, and $i < j$, then π' dominates π.
Rule 28. If $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $p^1_i = p^1_j$, $p^2_i = p^2_j$, and $d_i < d_j$, then π' dominates π.
Rule 29. If $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $p^1_i < p^1_j$, $p^2_i = p^2_j$, and $d_i < d_j$, then π' dominates π.
Rule 30. If $i \in \mathrm{AG}_1$, $j \in \mathrm{AG}_1$, $p^1_i = p^1_j$, $p^2_i < p^2_j$, and $d_i < d_j$, then π' dominates π.

Figure 2 shows a simple lower bound. The main idea behind the proposed lower bound is job preemption. Consider that the two machines fully parallel each other without any idle intervals. Then, the actual processing time of a job in such an ideal environment can be regarded as half the sum of its processing times on two machines. Therefore, we merge two processing times into one for each job and allocate the fabricated job to a virtual single machine (Steps 1–4). First, the two maintenance activities are scheduled first if they have not been scheduled in α yet. They are allocated to their respective maintenance windows as late as possible (Step 5). Then, For the n^β_2 remaining jobs from ag_2, we allocate them to the rear part of time interval $[0, M]$ in a preemptive way (Step 6). Finally, we sort the processing times and due dates of the n^β_1 remaining jobs for ag_1 in ascending order (Steps 7–8). Again, we allocate them to the schedule one by one in a preemptive way and finally return the estimated lower bound (Steps 9–11).

With the dominance rules and the lower bound, we are able to start to design the branch-and-bound algorithm (B&B) in a depth-first-search (DFS) manner. B&B includes the six major steps listed below. In Step 1, let B&B start with an approximate schedule ($\pi^\#$) provided by GA$^+$ (see the next section), and let it be the currently optimal schedule π^*. Note that $\pi^\#$ is organized to form a search tree and its root is at level 0. In the next two steps, we can prune some unnecessary nodes by the dominance rules and the lower bound since they definitely cannot achieve lower costs than the current one. In Step 4, if a leaf node is visited, evaluate its objective cost. If the root-to-leaf schedule achieves lower cost, replace the currently optimal solution with the better one. In Step 5, continue the search process in DFS order until all the remaining nodes are either visited or pruned. The last step returns the final results.

(1) **Initialization:** let $\pi^* = \pi^\#$, set $c^* = f(\pi^*)$, and start the search.
(2) **Branching:** for each node j at level k, construct the branches to the nodes at the next level $k+1$.
(3) **Bounding:** prune a subtree by applying the dominance rules and lower bound for the subtree rooted by node j.
(4) **Evaluation:** for a leaf node, replace π^* with this root-to-leaf schedule and reset c^* if it achieves a lower cost.
(5) **Termination:** recursively execute Steps 2–5 in DFS order until no more nodes can be visited.
(6) **Output:** if π^* is infeasible, then set $c^* = \infty$. Return π^* and c^*.

4. Genetic algorithm

For large problem instances, we propose a genetic algorithm to generate approximate schedules. For convenience, we name it GA$^+$. First, a greedy method is proposed to obtain a feasible solution. Second, a genetic algorithm is proposed and start with the feasible solution. Finally, GA$^+$ will generate an approximate solution $\pi^\#$.

First, to enhance the solution quality of GA$^+$, we propose a greedy method to generate a feasible initial schedule and will add it into the initial population of GA$^+$ later. The greedy method is shown in Figure 3. At the beginning, the jobs from ag_1 and ag_2 are sorted respectively (*i.e.*, Steps 1–2). We first satisfy constraints (2.4) and (2.5) by allocating the maintenance activities in Steps 4–5. Then we schedule the jobs from ag_2 as late as possible in order to satisfy constraint (2.3) in Step 6. Next, the jobs from ag_1 are scheduled one by one

```
Algorithm LowerBound
INPUT
    $(\alpha, \beta)$: $\alpha$ is the determined part and $\beta$ is the undetermined part
OUTPUT
    $LB$: lower bound
//Reorganize the copies of the jobs in $\beta$ and allocate the copies to a virtual single machine.
//i.e., obtain a new job (j) by merging two processing times of job j into one.-------------------
1)  Let $n_i^\beta$ be the number of jobs in $AG_i \cap \beta$ for $i = 1, 2, 3$;
2)  Set $p_{(j)} = 0.5(p_j^1 + p_j^2)$ and $d_{(j)} = d_j$ for $j = 1, 2, ..., n_1^\beta$;
3)  Set $p_{(j)} = 0.5(p_j^1 + p_j^2)$ for $j = 1, 2, ..., n_2^\beta$;
4)  Set $p_{(n-1)} = 0.5 p_{n-1}^1$ and $p_{(n)} = 0.5 p_n^2$;
//Schedule the maintenance activities from $ag_3$.--------------------------------------------------
5)  Allocate jobs (n-1) and (n) to $[a_{n-1}^1, b_{n-1}^1]$ and $[a_n^2, b_n^2]$ respectively as late as possible;
//Schedule the jobs from $ag_2$.----------------------------------------------------------------------
6)  Allocate the jobs from $ag_2$ in a preemptive way such that $\max\{C_{(j)}\} = M$;
//Schedule the jobs from $ag_1$.----------------------------------------------------------------------
7)  Sort all the $p_{(j)}$'s in ascending order; i.e., $p_{(1)} \le p_{(2)} \le ... \le p_{(n_1^\beta)}$;
8)  Sort all the $d_{(j)}$'s in ascending order; i.e., $d_{(1)} \le d_{(2)} \le ... \le d_{(n_1^\beta)}$;
9)  Allocate the fabricated jobs owned by $ag_1$ as early as possible in a preemptive way;
10) Compute the tardiness for each job owned by $ag_1$ and record the accumulative value in $LB$.
11) Return the estimated lower bound $LB$.
```

FIGURE 2. The proposed lower bound.

in Step 7. Finally, a feasible chromosome (*i.e.*, π^+) is obtained in Step 8 or we may temporarily assume there is no feasible chromosome in Step 9.

Second, a genetic algorithm (named GA$^+$) is proposed and its details are shown in Figure 4. In the initialization stage (*i.e.*, Step 1), let the population size be N and generate N chromosomes (*i.e.*, schedules) π_i for $i = 1, 2, \ldots, N$, where each chromosome is a random permutation of numbers $1, 2, \ldots, n$. Moreover, we replace the last chromosome with π^+ obtained by the previous greedy method if any.

In the evaluation stage (Steps 4–5), we calculate the objective cost of each chromosome in the current generation G. For each chromosome i, if some constraint is violated, a penalty $100\,n$ will be accumulatively inflicted on its objective cost; *i.e.*, $f(\pi_i)$. For simplicity, let $g_i = 1/(1 + f(\pi_i))$ be the fitness for each chromosome i. Meanwhile, let $\pi^\#$ be the currently best chromosome and record its lowest objective cost.

In the selection stage (Step 10), the standard roulette wheel [22] is employed to choose a parent chromosome, where the probability of selecting each chromosome i as a parent is $q_i = \sqrt{g_i}/\sum_{k=1}^{N}\sqrt{g_k}$. Like the Russian roulette game, each chromosome i has a probability of q_i to be chosen in the current generation. For each chromosome i in the current generation, we designate it as a parent. On the other hand, we employ the standard roulette wheel to choose some chromosome x to be another parent.

```
**Algorithm** *Greedy*
INPUT
    $p_j^1$, $p_j^2$, $M$, $[a_{n-1}^1, b_{n-1}^1]$, $[a_n^2, b_n^2]$: a problem instance
OUTPUT
    $\pi^+$: a feasible schedule
//Schedule the jobs from $ag_3$.----------------------------------------------
1)  Sort the jobs in $AG_1$ in ascending order of $d_j$;
2)  Sort the jobs in $AG_2$ in ascending order of $p_j^1$ - $p_j^2$;
3)  Let $t$ be a time table that is divided into time slices on the two machines;
4)  Allocate the maintenance activity $n-1$ to the rear part of $[a_{n-1}^1, b_{n-1}^1]$ in $t$ for machine 1;
5)  Allocate the maintenance activity $n$ to the rear part of $[a_n^2, b_n^2]$ in $t$ for machine 2;
6)  Allocate the jobs in $AG_2$ one by one to the time slices *backwards* from time $M$ in a non-
    preemptive way;
7)  Allocate the jobs in $AG_1$ one by one to the time slices forwards from time 1 in a non-
    preemptive way;
8)  Transcribe the allocated time slices in $t$ into a schedule $\pi^+$; i.e., a permutation of numbers 1,
    2, …, $n$;
9)  If the time slices are not enough for the jobs in $AG_2$, then set $c^+ = \infty$; else set $c^+ = f(\pi^+)$.
10) Return $\pi^+$
```

FIGURE 3. An initial schedule generated by a greedy method.

In the crossover stage (Step 11), the PMX crossover operation is conducted [12,22,32]. For each generation, about $R_{\text{crossover}}N$ new chromosomes will be generated by crossover and about $(1 - R_{\text{crossover}})N$ chromosomes will directly survive into the next generation. Figure 5 shows how parent i crossovers with parent x; *i.e.*, the PMX crossover. First, two distinct positions are randomly chosen to form two work areas (*i.e.*, shaded areas). Then we exchange the genes in the areas. There might be duplicate genes in the two parent chromosomes. To correct the errors, we need their corresponding mapping relationships such as Figure 5c. If some gene is duplicate in a chromosome, we repeatedly replace it with its neighbor(s) in its corresponding mapping relationship set until there is no duplicate gene in any chromosome.

In the mutation stage (Steps 12–14), we perform the extract-shift-insert mutation operation [22,26]. For each generation, about $R_{\text{mutate}}N$ chromosomes are randomly chosen to mutate. For each chosen chromosome, two positions are randomly selected to form a work area like the previous stage. We extract the leftmost gene in the work area. Then we left shift the other genes in the work area. Finally, we insert the extracted gene back into the rightmost position of the work area.

After the above four stages, a new generation is generated. The latter three stages will be repeated until some stopping criterion is met. That is, GA$^+$ will stop if it runs out of time or reaches the maximum generation limit. Finally, an approximate chromosome (*i.e.*, $\pi^\#$) is returned.

Lemma 4.1. *The maximum idle time on machine 1 is less than* $2\max\{p_j^1\}$*, and the maximum idle time on machine 2 is less than* $\max\{p_j^1\} + \max\{p_j^2\}$.

```
Algorithm $GA^+$
INPUT
    $\pi^+$ : a feasible schedule
    $N$ : the population size
    $R_{crossover}$ : the crossover rate
    $R_{mutate}$ : the mutation rate
    $T_{stop}$ , $G_{stop}$ : the stopping criteria
OUTPUT
    $\pi^\#$ : an approximate schedule
1)  Generate $N$ random chromosomes $\pi_i$ for $i=1,2,...,N$ and replace the last chromosome
    in the initial generation with $\pi^+$;
2)  Let $t=0$ and $G=0$;
3)  While $t<T_{stop}$ and $G<G_{stop}$ do Steps 4–15;
4)      Determine $f(\pi_i)$ for $i=1,2,...,N$;
5)      Let $\pi^\#$ be the best chromosome; i.e., $f(\pi^\#)\le f(\pi_i)$ for $i=1,2,...,N$;
7)      For $i=1,2,...,N$ do Steps 8–11;
8)          Set $r_1=rand()$;   //generate a random float number
9)          If $r_1<R_{crossover}$ then Steps 10–11;
10)             Choose a chromosome x with the standard roulette wheel selection;
11)             Perform the PMX crossover operation on chromosomes i and x;
12)     For $i=1,2,...,N$ do Steps 13–14;
13)         Set $r_2=rand()$;
14)         If $r_2<R_{mutate}$ then perform the mutation operation on chromosome i;
15)     Set $G=G+1$ and let $t$ be the current run time;
16) If $\pi^\#$ is infeasible then set $c^\#=\infty$;
17) Return $\pi^\#$.
```

FIGURE 4. The proposed genetic algorithm.

Proof. The main idea behind the lemma is to remove the influence of idle intervals and maintenance activities on the two machines. Consider the two continuous time axes on the two machines. To allocate maintenance activity $n-1$ on machine 1, we may cause an idle interval of p^1_{n-1} on machine 2. Moreover, there might be an extra idle interval before maintenance activity $n-1$ on machine 1 whose length is less than $\max\{p^1_j\}$ that cannot be utilized by other jobs. Therefore, we remark two intervals of length $\max\{p^1_j\}+p^1_{n-1}$ near a^1_{n-1} on the two time axes respectively; *i.e.*, $\max\{p^1_j\}+p^1_{n-1}\le 2\max\{p^1_j\}$. On the other hand, performing activity n on machine 2 will not delay any jobs on machine 1. But it might lead to an extra idle interval before maintenance activity n on machine 2 whose length is less than $\max\{p^1_j\}+\max\{p^2_j\}$ that cannot be utilized by other jobs. As a result, we remark two intervals of length $\max\{p^1_j\}+\max\{p^2_j\}$ near a^2_n on the two time axes respectively; *i.e.*, $\max\{p^1_j\}+p^2_n\le\max\{p^1_j\}+\max\{p^2_j\}$. □

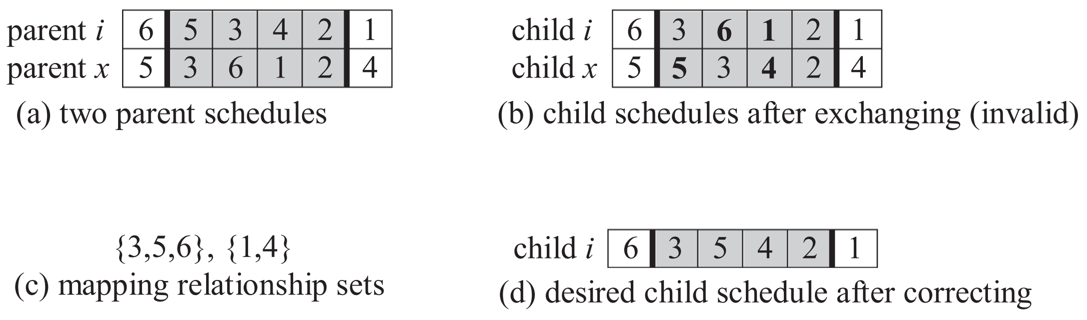

FIGURE 5. An example of the PMX crossover operation.

The above lemma can help us to estimate the maximum idle time on each machine. Note that the time difference between the two machines end their last jobs is not considered as idle time. This is because machine 1 has completed all its jobs and it can be shut down for other purposes (*e.g.*, maintenance) instead of wasting time and energy. Now we treat the two remaining time axes as two continuous axes and allocate jobs in AG_1 and AG_2 without the interference of maintenance activities. That is, if some job is allocated to some remarked area, let it be. Its subsequent job needs to escape the remarked area and starts immediately after the remarked area. The jobs in AG_2 are first allocated at the bottoms of $[0,M]$ on the two axes. Then all the jobs in AG_1 are allocated in EDD order. The corresponding job schedule can be determined easily. The only difference to the physical optimal schedule is that the two remarked time intervals might lead to some wasted idle time. Consequently, the relative error bound is $n_1(3\max\{p_j^1\}+\max\{p_j^2\})/n$.

5. Experimental results

Before conducting experiments, we list all the parameters used in the related experiments in Table 1. Parameter n (*i.e.*, the total number of jobs from three agents) has been defined in Section 2, where $n = n_1 + n_2 + n_3$. We assume that parameters p_j^1 and p_j^2 follow a discrete uniform distribution $\mathrm{DU}(1,100)$ and that parameter d_j follows a discrete uniform distribution $\mathrm{DU}(\sum(p_j^1+p_j^2)(1-\tau-R/2),\sum(p_j^1+p_j^2)(1-\tau+R/2))$, where τ and R are two control parameters. The smaller R is, the more centralized due dates we have; the larger τ is, the earlier due dates we have. Parameter M follows a discrete uniform distribution $\mathrm{DU}(0,10\,n)$. Moreover, the related parameters regarding the genetic algorithm are the same as those defined in the previous section. All the algorithms are implemented in Object Pascal and executed in a Windows 7 environment on an Intel Core i7 3770@3.40 GHz with 8G RAM. Later, each experiment is controlled by the above parameters. For each parameter setting, 50 instances are generated, and their experimental results are recorded and analyzed.

The experiments can be divided into four parts. First, we conduct pilot experiments to determine the default settings for the following two genetic algorithms. Let GA denote the original genetic algorithms described in the previous section and GA^+ the improved genetic algorithm. For GA^+, an initial chromosome obtained by the greedy method shown in Figure 3 is added into the first generation. Moreover, for each following generation, 100 extra mutations are conducted on the currently best chromosome to test if GA^+ can improve the solution quality further. Now we observe how the crossover rate and mutation rate influence the solution quality and execution speed. Figure 6 shows that GA^+ always achieves satisfactory solution quality when $R_{\mathrm{crossover}} = 0.8$. The relative error is defined as $(c_c^*)/n$, where c means the objective cost obtained by a genetic algorithm and c^* means the optimal objective cost obtained by B&B. It is clear that the relative error is almost zero when $R_{\mathrm{crossover}} = 0.8$. On the other hand, Figure 7 shows different run times for different mutation rates. In general, $R_{\mathrm{mutate}} = 0.4$ is a medium setting that will not cause much run time. Moreover, $R_{\mathrm{mutate}} = 0.4$ can achieve better diversity than other settings. Therefore, we let $R_{\mathrm{crossover}} = 0.8$ and $R_{\mathrm{mutate}} = 0.4$ be the default settings in the following experiments. Note that throughout this section, we let the population size $N = 500$ and force

Table 1. The parameter settings.

Parameter	Default value	Range	Meaning
n		12, 14, 50, 100	Number of jobs
n_i			Number of jobs agent ag_i
AG_i			Set of jobs from agent ag_i
p_j^1		$1, 2, \ldots, 100$	Processing time of job j on machine 1
p_j^2		$1, 2, \ldots, 100$	Processing time of job j on machine 2
d_j		Controlled by τ and R	Due data of job j
τ	0.5	0.25, 0.5, 0.75	Control parameter for due date
R	0.5	0.25, 0.5, 0.75	Control parameter for due date
M			Time limit for AG_2
N	500		Population size
$[a_{n-1}^1, b_{n-1}^1]$			Maintenance window for job $n-1$ on machine 1
$[a_n^2, b_n^2]$			Maintenance window for job n on machine 2
$R_{\mathrm{crossover}}$	0.8		Crossover rate
R_{mutate}	0.4		Mutation rate
G_{stop}	500		Stopping criterion (500 generations)
T_{stop}	n		Stopping criterion (run time in s)

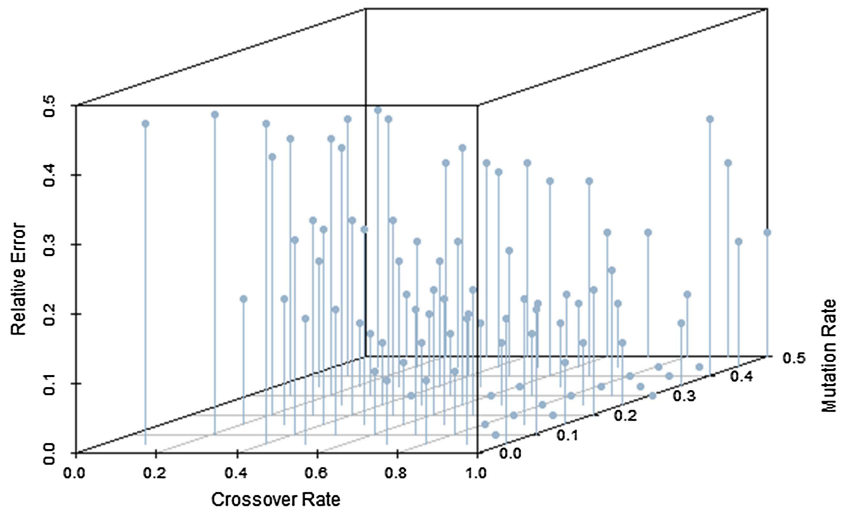

Figure 6. The solution quality under different parameter settings.

a genetic algorithm to stop if no improvement can be made during the most recent 500 generations or the run time exceeds n seconds.

Second, we compare the experimental results of the three algorithms to compare their solution qualities and execution speeds. In Table 2, there are 12 jobs, and each run time is measured in seconds. For B&B, it will take more run time even if we have only a few jobs from ag_1; *i.e.*, a small n_1. In Table 3, B&B needs more run time. For some instances with earlier due dates and fewer jobs from ag_1, B&B even takes 1523.46 s to solve an instance on average. Note that column NI denotes the number of insolvable instances (including the instances that cannot be solved within a billion nodes). Note that there exist insolvable instances and they are viewed as an inevitable side effect. It is intuitive to avoid generating such insolvable instances and hence experiments can be easily and quickly conducted. However, it would be a dilemma. If we relax the control over parameter

TABLE 2. The comparison of two genetic algorithms for $n = 12$.

					B&B					GA					GA$^+$				
					Nodes		Run time (s)			Run time (s)		Relative error			Run time (s)		Relative error		
τ	R	n_1	n_2	n_3	Mean	Max	Mean	Max	NI	Mean	Max	Mean	Max	NI	Mean	Max	Mean	Max	NI
0.25	0.25	2	8	2	1 406 509	3 270 131	0.26	8.41	18	0.08	1.95	0.00	0.00	18	0.08	1.95	0.00	0.00	18
	0.50				2 754 718	8 738 727	1.27	21.51	22	0.24	2.00	0.00	0.00	22	0.23	1.97	0.00	0.00	22
	0.75				3 553 109	11 994 857	4.48	31.68	18	0.69	2.01	0.16	5.17	18	0.63	2.00	0.00	0.00	18
0.50	0.25				5 160 770	22 464 774	9.41	53.71	20	1.41	2.47	0.21	5.83	20	1.52	3.10	0.19	5.83	20
	0.50				3 929 191	13 046 208	7.50	34.18	20	1.24	2.22	0.06	1.83	20	1.30	3.79	0.00	0.00	20
	0.75				4 929 805	20 992 406	10.14	52.79	18	1.37	2.50	0.00	0.00	18	1.37	2.47	0.71	22.17	18
0.75	0.25				5 869 905	16 372 320	16.20	37.82	16	2.01	2.37	0.00	0.00	17	2.08	3.43	1.25	42.42	16
	0.50				5 935 143	21 911 955	16.04	52.96	14	1.99	3.18	2.00	33.17	19	2.00	3.42	1.88	33.17	19
0.25	0.25	4	6	2	1 059 505	1 869 570	0.14	4.01	22	0.09	1.95	0.00	0.00	22	0.09	1.98	0.00	0.00	22
	0.50				690 561.2	1 579 418	0.00	0.00	13	0.03	0.11	0.00	0.00	13	0.03	0.08	0.00	0.00	13
	0.75				1 280 589	4 247 374	1.52	11.62	12	0.60	2.06	0.00	0.00	12	0.60	2.01	0.00	0.00	12
0.50	0.25				1 842 132	5 738 034	5.01	14.74	19	1.79	5.26	0.00	0.00	19	1.72	2.61	0.03	0.92	19
	0.50				1 441 829	5 694 842	3.93	14.01	18	1.70	2.23	0.56	16.75	20	1.68	2.34	0.56	16.75	20
	0.75				2 136 742	4 362 513	4.66	10.98	21	1.69	2.39	0.68	18.58	21	1.61	2.26	0.03	1.00	21
0.75	0.25				2 939 509	15 294 346	8.59	34.55	19	2.04	2.53	0.99	20.25	19	2.06	2.81	0.50	10.17	19
	0.50				1 846 843	3 907 621	5.97	11.69	15	2.12	3.09	0.93	31.00	16	2.07	2.68	0.93	31.00	16
0.25	0.25	6	4	2	1 144 676	2 769 898	0.00	0.00	19	0.02	0.03	0.00	0.00	19	0.02	0.05	0.00	0.00	19
	0.50				707 962.1	1 612 775	0.00	0.00	18	0.02	0.05	0.00	0.00	18	0.02	0.03	0.00	0.00	18
	0.75				1 138 247	3 481 313	0.28	9.06	17	0.08	1.95	0.00	0.00	17	0.07	1.95	0.00	0.00	17
0.50	0.25				1 747 465	4 045 193	3.14	10.39	18	1.24	3.40	0.03	0.92	18	1.14	2.11	0.22	6.33	18
	0.50				1 698 313	4 425 703	2.88	10.62	20	1.16	2.23	0.00	0.00	20	1.15	2.08	0.12	3.58	20
	0.75				1 603 418	5 117 513	4.12	12.76	19	1.73	3.12	0.76	14.50	19	1.70	2.39	0.29	9.08	19
0.75	0.25				1 756 205	3 982 071	4.96	11.01	16	2.35	5.77	2.10	28.08	16	2.21	3.56	1.03	28.08	16
	0.50				2 186 424	4 798 205	6.11	12.22	14	2.26	3.68	0.24	4.25	14	2.18	3.99	1.68	40.75	14

TABLE 3. The comparison of two genetic algorithms $n = 14$.

					B&B					GA					GA$^+$				
					Nodes		Run time (s)			Run time (s)		Relative error			Run time (s)		Relative error		
τ	R	n_1	n_2	n_3	Mean	Max	Mean	Max	NI	Mean	Max	Mean	Max	NI	Mean	Max	Mean	Max	NI
0.25	0.25	2	10	2	161 555 157	429 646 738	0.00	0.00	19	0.04	0.17	0.00	0.00	19	0.03	0.11	0.00	0.00	19
	0.50				108 644 455	330 387 217	44.80	1025.43	22	0.21	2.34	0.00	0.00	22	0.22	2.67	0.00	0.00	22
	0.75				186 554 762	502 642 192	211.78	1556.42	16	0.66	2.92	0.49	11.00	16	0.54	2.54	0.00	0.00	16
0.50	0.25				364 353 248	765 431 883	808.08	2086.19	30	1.64	2.53	0.14	2.86	30	1.70	2.76	0.73	14.14	30
	0.50				325 079 429	901 632 617	633.81	2359.52	14	1.40	2.53	0.00	0.14	14	1.41	2.93	0.00	0.00	14
	0.75				286 124 667	758v265 894	589.31	2122.25	19	1.50	3.95	0.03	0.86	19	1.56	4.27	1.00	30.29	19
0.75	0.25				564 989 920	966 770 005	1523.46	2878.92	20	2.19	2.57	1.97	30.86	22	2.17	4.45	1.72	30.86	22
	0.50				415 168 680	922 175 068	1188.51	3107.57	21	2.16	2.48	0.03	0.71	24	2.18	2.56	1.43	35.64	24
0.25	0.25	4	8	2	47 614 114	254 502 623	6.20	126.78	17	0.05	0.16	0.00	0.00	19	0.04	0.08	0.00	0.00	19
	0.50				80 570 465	330 770 072	64.96	878.10	23	0.21	2.29	0.00	0.00	23	0.21	2.33	0.00	0.00	23
	0.75				67 339 664	183 609 280	47.24	537.22	16	0.38	2.42	0.07	2.50	16	0.40	2.47	0.01	0.43	16
0.50	0.25				84 544 593	630 910 696	231.78	1556.67	19	1.95	3.25	0.48	9.29	20	1.96	2.98	0.39	9.29	20
	0.50				98 477 910	435 242 712	246.87	1148.89	12	1.71	3.64	0.02	0.57	18	1.78	4.56	0.25	5.64	19
	0.75				81 234 423	231 203 878	237.01	728.18	18	1.74	3.82	0.00	0.00	20	1.76	3.46	0.00	0.00	20
0.75	0.25				234 695 072	824 904 514	741.51	2373.29	15	2.53	4.29	1.31	13.86	15	2.63	5.09	2.01	44.36	15
	0.50				215 953 990	544 430 638	687.29	1618.01	21	2.66	5.15	1.63	17.79	21	2.62	4.15	1.12	16.00	21
0.25	0.25	6	6	2	24 382 440	84 968 824	0.00	0.00	17	0.03	0.09	0.00	0.00	17	0.03	0.09	0.00	0.00	17
	0.50				32 035 743	52 606 235	0.00	0.00	10	0.03	0.08	0.00	0.00	10	0.03	0.11	0.00	0.00	10
	0.75				46 658 039	166 235 088	22.61	436.27	12	0.29	2.53	0.00	0.07	12	0.29	2.56	0.00	0.07	12
0.50	0.25				66 418 346	181 927 545	130.37	507.66	25	1.18	2.62	0.73	18.36	25	1.20	2.57	0.81	18.36	25
	0.50				60 791 040	184 787 024	161.79	511.12	17	1.68	3.56	0.36	12.00	17	1.72	3.42	0.45	12.00	17
	0.75				74 681 787	284 213 314	193.19	786.09	13	1.90	4.06	0.64	15.50	14	1.95	4.57	0.04	1.50	14
0.75	0.25				72 162 950	220 022 172	234.66	637.19	7	2.59	4.17	1.70	18.57	7	2.65	3.74	1.71	28.64	7
	0.50				130 016 373	988 834 987	387.02	2509.15	15	2.79	4.79	3.53	39.29	16	2.71	4.45	3.88	31.07	16

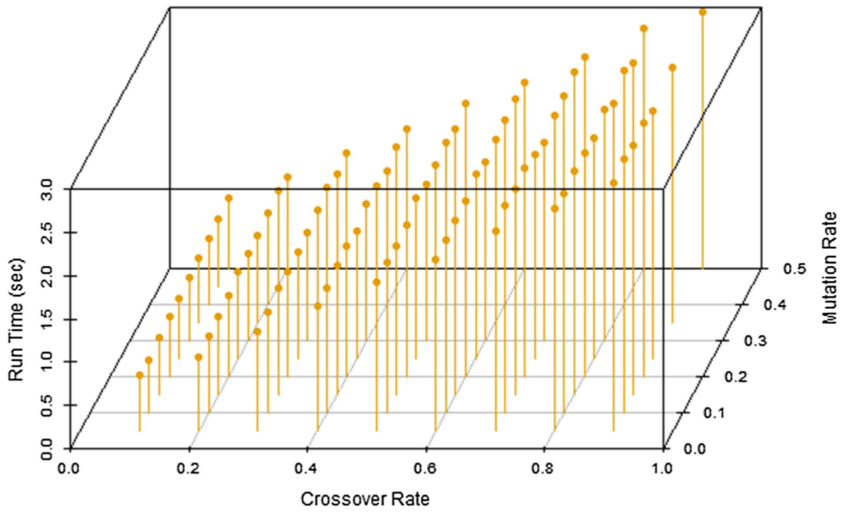

FIGURE 7. The execution efficiency under different parameter settings.

settings, problem instances will be easy and can be solved in a short time; nevertheless, the discrimination between different algorithms will be vague. On the other hand, if we insist on generating fair and comprehensive problem instances, some instances may be difficult or even insolvable; however, these difficult instances are useful to test the abilities of metaheuristic algorithms. Consequently, we require retaining these difficult instances to evaluate metaheuristic algorithms and reluctantly tolerate this side effect. Especially when the problem size is large, the requirement becomes more obvious. In Tables 2 and 3, the run times of GA$^+$ and GA are similar. Unlike B&B, the run times of both genetic algorithms increase slightly when the problem size increases. On the other hand, GA$^+$ usually outperforms GA in terms of solution quality. This happens because GA$^+$ utilizes a feasible chromosome at the beginning. Moreover, the solution quality is slightly decreasing when we have a large n_1 and a large τ. It means that GA$^+$ takes more trials to locate the global optimums if there are more jobs from ag_1 and all of them have earlier due dates.

Third, two experiments are conducted to observe the performance of metaheuristic algorithms when the problem size is large. Since B&B cannot provide the optimal solutions when the problem size is large, we compare the two genetic algorithms only. Tables 4 and 5 show the performance of both genetic algorithms. The relative deviation is defined as $(c-l)/c$, where c means the objective cost obtained by two genetic algorithms and l means the proposed lower bound. If $c=0$ and $l=0$, we let the relative deviation be 0. On the other hand, for the case $l=0$ and $c\neq 0$, the relative deviation is denoted by "–". Again, for the difficult instances ($i.e.$, large n_1 or large τ), both algorithms can take the predefined maximum run time ($i.e.$, n seconds) only, where n is the number of jobs. As shown in the two tables, in general, GA$^+$ leaves fewer instances insolvable and outperforms GA in terms of solution quality. Therefore, GA$^+$ achieves better stability. This is because GA$^+$ has a feasible solution at the beginning of evolution. If the mutation rate and crossover rate are specified properly, GA$^+$ is taught to obtain another better solution if any. Therefore, the final objective cost is very likely to be lower than the initial one. Conversely, GA has no such safety mechanisms and its solution quality is naturally uncertain.

Fourth, Figure 8 shows the results of sensitivity analyses on different parameters. When we adjust the magnitude of a parameter from -15% to 15%, the default values of other parameters are $n=12$, $n_1=6$, $M=0.75\sum_{j\in\mathrm{AG}_1}(p_j^1+p_j^2)$, $\tau=0.5$, and $R=0.5$. For each setting, we conduct 50 random trials. There might be some insolvable instances for each setting; $i.e.$, constraints are too strict. For processing time, it is the only parameter that varies positively. The objective cost increases quickly if p_j increases. Conversely, if we relax the parameter of due date (d_j), we have a larger freedom to schedule jobs and thus have lower objective costs. On

Table 4. The comparison of two genetic algorithms for $n = 50$.

					GA					GA$^+$				
					Run time (s)		Relative deviation			Run time (s)		Relative deviation		
τ	R	n_1	n_2	n_3	Mean	Max	Mean	Max	NI	Mean	Max	Mean	Max	NI
0.25	0.25	10	38	2	0.82	3.31	0.000	0.000	6	0.85	3.56	0.000	0.000	6
	0.50				0.79	4.03	0.000	0.000	3	0.69	2.53	0.000	0.000	3
	0.75				1.77	11.65	0.071	–	8	1.89	14.02	0.093	–	7
0.50	0.25				6.54	25.96	0.409	–	6	6.42	20.25	0.409	–	6
	0.50				8.76	28.03	0.614	–	3	9.87	26.55	0.614	–	3
	0.75				8.43	27.52	0.497	–	4	7.59	21.84	0.497	–	4
0.75	0.25				14.52	36.47	0.746	–	2	15.16	30.48	0.730	–	2
	0.50				17.81	36.16	0.871	–	2	17.46	43.32	0.868	–	2
0.25	0.25	20	28	2	1.68	4.54	0.000	0.000	4	1.71	4.60	0.000	0.000	4
	0.50				1.51	3.68	0.000	0.000	6	1.42	5.54	0.000	0.000	6
	0.75				1.98	10.39	0.000	0.000	1	1.76	5.87	0.000	0.000	1
0.50	0.25				2.52	17.14	0.043	–	3	2.38	17.36	0.043	–	3
	0.50				10.88	34.68	0.522	–	4	10.87	41.68	0.489	–	3
	0.75				15.68	39.06	0.694	–	3	13.31	38.38	0.694	–	3
0.75	0.25				32.66	50.05	0.501	0.708	4	33.75	50.03	0.473	0.687	4
	0.50				36.90	50.05	0.772	–	2	36.11	50.03	0.751	–	2
0.25	0.25	30	18	2	1.35	2.89	0.000	0.000	1	1.31	2.70	0.000	0.000	1
	0.50				1.36	3.53	0.000	0.000	2	1.27	2.68	0.000	0.000	2
	0.75				1.34	2.36	0.000	0.000	3	1.34	2.65	0.000	0.000	3
0.50	0.25				1.68	3.04	0.000	0.000	5	1.58	2.59	0.000	0.000	5
	0.50				2.85	22.40	0.061	–	1	2.83	17.75	0.061	–	1
	0.75				12.89	38.86	0.617	–	3	11.73	33.82	0.617	–	3
0.75	0.25				40.92	50.05	0.526	0.767	2	43.79	50.05	0.499	0.760	2
	0.50				42.96	50.05	0.808	–	0	44.22	50.05	0.788	–	1

the other hand, it is interesting that objective cost varies insignificantly with maintenance window size $(b_i - a_i)$ and completion time limit (M). Although M does not affect objective cost greatly, a large value of M implies fewer insolvable instances (*i.e.*, NI). It is also another kind of positive influence on objective cost. Moreover, larger maintenance windows do not imply lower objective costs. Therefore, we can set $b^1_{n-1} - a^1_{n-1} = 2p^1_{n-1}$ and $b^2_n - a^2_n = 2p^2_n$ to schedule jobs; *i.e.*, it is wide enough. It is helpful for us to determine the widths of maintenance windows.

6. Conclusion

This study investigates a three-agent flowshop problem on two machines. Agent 1 aims to minimize its total tardiness subject to the constraints that the maximum completion time of agent 2's jobs cannot exceed a fixed limit and that agent 3 must complete its two maintenance activities within two maintenance windows. A branch-and-bound algorithm (B&B) and an improved genetic algorithm (GA$^+$) are proposed. In general, due to the nature of a genetic algorithm (*i.e.*, random walk), the solution quality is usually unstable. In this study, consequently, we hybridize a feasible chromosome into the initial population. That is, we reduce the adverse influence of the solutions randomly generated in the initial population. As shown in these experiments, the maximum mean of relative errors can be reduced to 3.88 or lower when the problem size is small. It implies that hybridizing a feasible solution is helpful to improve the solution quality of GA$^+$.

However, when the problem size is large, it is difficult to obtain the optimal solutions for evaluating the performance of GA$^+$. Consequently, we propose a relative deviation to show the gap between GA$^+$ and another

TABLE 5. The comparison of two genetic algorithms for $n = 100$.

					GA					GA$^+$				
					Run time (s)		Relative deviation			Run time (s)		Relative deviation		
τ	R	n_1	n_2	n_3	Mean	Max	Mean	Max	NI	Mean	Max	Mean	Max	NI
0.25	0.25	20	78	2	7.18	37.39	0.000	0.000	2	7.79	50.78	0.000	0.000	2
	0.50				6.49	32.36	0.000	0.000	2	6.07	23.62	0.000	0.000	2
	0.75				8.87	53.54	0.063	–	2	8.91	47.00	0.063	–	2
0.50	0.25				31.81	100.06	0.435	–	5	28.80	90.61	0.435	–	4
	0.50				33.68	100.04	0.550	–	1	34.57	100.09	0.529	–	1
	0.75				33.82	100.06	0.445	–	1	30.88	100.03	0.446	–	1
0.75	0.25				76.41	100.08	0.684	–	2	76.76	100.09	0.658	–	2
	0.50				79.60	100.09	0.805	–	0	79.93	100.08	0.785	–	0
0.25	0.25	40	58	2	14.68	42.07	0.000	0.000	4	13.91	37.74	0.000	0.000	3
	0.50				16.25	50.55	0.000	0.000	2	13.85	44.21	0.000	0.000	2
	0.75				15.64	40.54	0.000	0.000	1	15.12	38.11	0.000	0.000	1
0.50	0.25				17.00	44.46	0.000	0.000	3	15.66	33.99	0.000	0.000	4
	0.50				44.63	100.08	0.500	–	0	42.85	100.08	0.500	–	0
	0.75				62.34	100.08	0.608	–	2	60.66	100.08	0.607	–	2
0.75	0.25				99.87	100.09	0.493	0.641	13	100.04	100.08	0.432	0.618	9
	0.50				100.04	100.08	0.817	0.996	11	100.04	100.09	0.773	0.993	9
0.25	0.25	60	38	2	11.51	20.56	0.000	0.000	1	11.36	24.09	0.000	0.000	1
	0.50				11.91	27.94	0.000	0.000	1	11.22	19.02	0.000	0.000	1
	0.75				12.31	30.97	0.000	0.000	0	10.74	21.15	0.000	0.000	0
0.50	0.25				12.88	27.07	0.000	0.000	2	11.86	19.91	0.000	0.000	2
	0.50				15.02	22.59	0.000	0.000	4	14.33	22.40	0.000	0.000	4
	0.75				58.44	100.09	0.617	–	2	56.50	100.08	0.626	–	2
0.75	0.25				100.04	100.08	0.481	0.611	26	100.05	100.08	0.435	0.575	15
	0.50				100.05	100.09	0.832	–	21	100.05	100.09	0.778	–	16

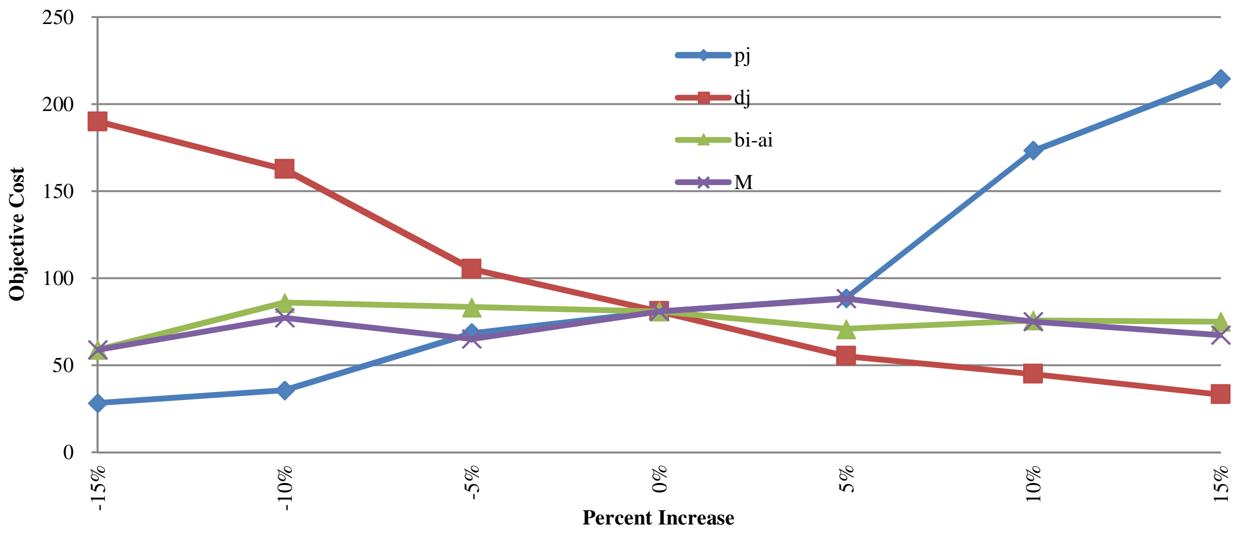

FIGURE 8. The effects of input parameters on the objective cost.

ordinary genetic algorithm. Moreover, when we generate large and difficult problem instances, some insolvable instances will emerge. It is also worthwhile to explore how to generate adequate instances as well as avoid insolvable instances. In the future, we design a more powerful lower bound or a more efficient crossover operator to deal with these difficult instances.

Acknowledgements. This study was supported under Contract NSC 103-2221-E-035-075-MY3. The authors thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this study.

References

[1] A. Agnetis, G. de Pascale and D. Pacciarelli, A Lagrangian approach to single-machine scheduling problems with two competing agents. *J. Scheduling* **12** (2009) 401–415.

[2] F. Ahmadizar and J. Eteghadipour, Single-machine earliness-tardiness scheduling with two competing agents and idle time. *Eng. Optim.* **49** (2017) 499–512.

[3] K.R. Baker and J.C. Smith, A multiple-criterion model for machine scheduling. *J. Scheduling* **6** (2003) 7–16.

[4] S. Bouzidi-Hassini, F.B.S. Tayeb, F. Marmier and M. Rabahi, Considering human resource constraints for real joint production and maintenance schedules. *Comput. Ind. Eng.* **90** (2015) 197–211.

[5] X.L. Cao, W.H. Wu, W.H. Wu and C.C. Wu, Some two-agent single-machine scheduling problems to minimize minmax and minsum of completion times. *Oper. Res.* **18** (2018) 293–314.

[6] W.J. Chen, Scheduling of jobs and maintenance in a textile company. *Int. J. Adv. Manuf. Technol.* **31** (2007) 737–742.

[7] W.J. Chen, Scheduling with dependent setups and maintenance in a textile company. *Comput. Ind. Eng.* **57** (2009) 867–873.

[8] T.C.E. Cheng, C.Y. Liu, W.C. Lee, M. Ji, Two-agent single-machine scheduling to minimize the weighted sum of the agents' objective functions. *Comput. Ind. Eng.* **78** (2014) 66–73.

[9] M.B. Cheng, P.R. Tadikamalla, J. Shang and B.X. Zhang, Two-machine flow shop scheduling with deteriorating jobs: minimizing the weighted sum of makespan and total completion time. *J. Oper. Res. Soc.* **66** (2015) 709–719.

[10] S.R. Cheng, Y.Q. Yin, C.H. Wen, W.C. Lin, C.C. Wu and J. Liu, A two-machine flowshop scheduling problem with precedence constraint on two jobs. *Soft Comput.* **21** (2017) 2091–2103.

[11] S. Gawiejnowicz, W.C. Lee, C.L. Lin and C.C. Wu, Single-machine scheduling of proportionally deteriorating jobs by two agents. *J. Oper. Res. Soc.* **62** (2011) 1983–1991.

[12] D.E. Goldberg and R. Lingle, Alleles, loci and the traveling salesman problem. In: Proceedings of an International Conference on Genetic Algorithms and Their Application, Hillsdale, NJ, USA (1985).

[13] L. Grigoriu and D. Briskorn, Scheduling jobs and maintenance activities subject to job-dependent machine deteriorations. *J. Scheduling* **20** (2017) 183–197.

[14] M.Z. Gu, J.W. Gu and X.W. Lu, An algorithm for multi-agent scheduling to minimize the makespan on m parallel machines. *J. Scheduling* **21** (2018) 483–492.

[15] M. Haouari and M. Kharbeche, An assignment-based lower bound for a class of two-machine flow shop problems. *Comput. Oper. Res.* **40** (2013) 1693–1699.

[16] R. Jamshidi and M.M.S. Esfahani, Reliability-based maintenance and job scheduling for identical parallel machines. *Int. J. Prod. Res.* **53** (2015) 1216–1227.

[17] B. Jeong and Y.D. Kim, Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times, *Comput. Oper. Res.*, **47** (2014), 72-80.

[18] V. Kachitvichyanukul, Comparison of three evolutionary algorithms: GA, PSO, and DE. *Ind. Eng. Manage. Syst.* **11** (2012) 215–223.

[19] Y.D. Kim, A new branch and bound algorithm for minimizing mean tardiness in 2-machine flowshops. *Comput. Oper. Res.* **20** (1993) 391–401.

[20] J.Y. Lee and Y.D. Kim, A branch and bound algorithm to minimize total tardiness of jobs in a two identical-parallel-machine scheduling problem with a machine availability constraint. *J. Oper. Res. Soc.* **66** (2015) 1542–1554.

[21] W.C. Lee and J.Y. Wang, A scheduling problem with three competing agents. *Comput. Oper. Res.* **51** (2014) 208–217.

[22] W.C. Lee and J.Y. Wang, A three-agent scheduling problem for minimizing the makespan on a single machine. *Comput. Ind. Eng.* **106** (2017) 147–160.

[23] W.C. Lee and C.C. Wu, Minimizing the total flow time and the tardiness in a two-machine flow shop. *Int. J. Syst. Sci.* **32** (2001) 365–373.

[24] W.C. Lee, S.K. Chen and C.C. Wu, Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem. *Expert Syst. App.* **37** (2010) 6594–6601.

[25] W.C. Lee, Y.H. Chung and M.C. Hu, Genetic algorithms for a two-agent single-machine problem with release time. *Appl. Soft Comput.* **12** (2012) 3580–3589.

[26] W.C. Lee, J.Y. Wang and H.W. Su, Algorithms for single-machine scheduling to minimize the total tardiness with learning effects and two competing agents. *Concurrent Eng.-Res. App.* **23** (2015) 13–26.

[27] W.C. Lee, J.Y. Wang and M.C. Lin, A branch-and-bound algorithm for minimizing the total weighted completion time on parallel identical machines with two competing agents. *Knowl.-Based Syst.* **105** (2016) 68–82.

[28] D.M. Lei, Variable neighborhood search for two-agent flow shop scheduling problem. *Comput. Ind. Eng.* **80** (2015) 125–131.
[29] B.M.T. Lin, F.J. Hwang and J.N.D. Gupta, Two-machine flowshop scheduling with three-operation jobs subject to a fixed job sequence. *J. Scheduling* **20** (2017) 293–302.
[30] P. Liu, N. Yi, X.Y. Zhou and H. Gong, Scheduling two agents with sum-of-processing-times-based deterioration on a single machine. *Appl. Math. Comput.* **219** (2013) 8848–8855.
[31] M. Liu, S.J. Wang, C.B. Chu and F. Chu, An improved exact algorithm for single-machine scheduling to minimise the number of tardy jobs with periodic maintenance. *Int. J. Prod. Res.* **54** (2016) 3591–3602.
[32] E.G. Lopez and M. O'Neill, On the effects of locality in a permutation problem: the Sudoku puzzle. In: IEEE Symposium on Computational Intelligence and Games. Milano, Italy (2009) 80–87.
[33] S.A. Mansouri and E. Aktas, Minimizing energy consumption and makespan in a two-machine flowshop scheduling problem. *J. Oper. Res. Soc.* **67** (2016) 1382–1394.
[34] B. Mor and G. Mosheiov, Minimizing maximum cost on a single machine with two competing agents and job rejection. *J. Oper. Res. Soc.* **67** (2016) 1524–1531.
[35] Y.D. Ni and Z.J. Zhao, Two-agent scheduling problem under fuzzy environment. *J. Intell. Manuf.* **28** (2017) 739–748.
[36] J.C.H. Pan, J.S. Chen and C.M. Chao, Minimizing tardiness in a two-machine flow-shop. *Comput. Oper. Res.* **29** (2002) 869–885.
[37] K. Rustogi and V.A. Strusevich, Single machine scheduling with time-dependent linear deterioration and rate-modifying maintenance, *J. Oper. Res. Soc.* **66** (2015) 500–515.
[38] J. Schaller, Note on minimizing total tardiness in a two-machine flowshop. *Comput. Oper. Res.* **32** (2005) 3273–3281.
[39] D. Shabtay, O. Dover and M. Kaspi, Single-machine two-agent scheduling involving a just-in-time criterion. *Int. J. Prod. Res.* **53** (2015) 2590–2604.
[40] Y.R. Shiau, W.C. Lee, Y.S. Kung and J.Y. Wang, A lower bound for minimizing the total completion time of a three-agent scheduling problem. *Inf. Sci.* **340** (2016) 305–320.
[41] J.M.P. Siopa, J.E.S. Garcao and J.M.E. Silva, Component redundancy allocation in optimal cost preventive maintenance scheduling. *J. Oper. Res. Soc.* **66** (2015) 925–935.
[42] L.H. Su and H.M. Wang, Minimizing total absolute deviation of job completion times on a single machine with cleaning activities. *Comput. Ind. Eng.* **103** (2017) 242–249.
[43] K. Thornblad, A.B. Stromberg, M. Patriksson and T. Almgren, Scheduling optimisation of a real flexible job shop including fixture availability and preventive maintenance. *Eur. J. Ind. Eng.* **9** (2015) 126–145.
[44] M. Torkashvand, B. Naderi and S.A. Hosseini, Modelling and scheduling multi-objective flow shop problems with interfering jobs. *Appl. Soft Comput.* **54** (2017) 221–228.
[45] J.Y. Wang, A branch-and-bound algorithm for minimizing the total tardiness of a three-agent scheduling problem considering the overlap effect and environmental protection. *IEEE Access* **7** (2019) 5106–5123.
[46] J.Y. Wang, Minimizing the total weighted tardiness of overlapping jobs on parallel machines with a learning effect. *J. Oper. Res. Soc.* Accepted (2019) https://doi.org/10.1080/01605682.2019.1590511.
[47] D.J. Wang, Y.Q. Yin, J.Y. Xu, W.H. Wu, S.R. Cheng and C.C. Wu, Some due date determination scheduling problems with two agents on a single machine. *Int. J. Prod. Econ.* **168** (2015) 81–90.
[48] J.Q. Wang, G.Q. Fan, Y.Q. Zhang, C.W. Zhang and J.Y.T. Leung, Two-agent scheduling on a single parallel-batching machine with equal processing time and non-identical job sizes. *Eur. J. Oper. Res.* **258** (2017) 478–490.
[49] D.J. Wang, Y.Q. Yin, W.H. Wu, W.H. Wu, C.C. Wu and P.H. Hsu, A two-agent single-machine scheduling problem to minimize the total cost with release dates. *Soft Comput.* **21** (2017) 805–816.
[50] W.H. Wu, Y.Q. Yin, T.C.E. Cheng, W.C. Lin, J.C. Chen, S.Y. Luo and C.C. Wu, A combined approach for two-agent scheduling with sum-of-processing-times-based learning effect. *J. Oper. Res. Soc.* **68** (2017) 111–120.
[51] Z.J. Xu and D.H. Xu, Single-machine scheduling with preemptive jobs and workload-dependent maintenance durations. *Oper. Res.* **15** (2015) 423–436.
[52] C.N. Yang, B.M.T. Lin, F.J. Hwang and M.C. Wang, Acquisition planning and scheduling of computing resources. *Comput. Oper. Res.* **76** (2016) 167–182.
[53] J.F. Ye and H.M. Ma, Multiobjective joint optimization of production scheduling and maintenance planning in the flexible job-shop problem. *Math. Probl. Eng.* **2015** (2015) 725460.
[54] Y.Q. Yin, C.C. Wu, W.H. Wu, C.J. Hsu and W.H. Wu, A branch-and-bound procedure for a single-machine earliness scheduling problem with two agents. *Appl. Soft Comput.* **13** (2013) 1042–1054.
[55] Y.Q. Yin, D.S. Ye and G.C. Zhang, Single machine batch scheduling to minimize the sum of total flow time and batch delivery cost with an unavailability interval. *Inf. Sci.* **274** (2014) 310–322.
[56] Y.Q. Yin, Y. Wang, T.C.E. Cheng, D.J. Wang and C.C. Wu, Two-agent single-machine scheduling to minimize the batch delivery cost. *Comput. Ind. Eng.* **92** (2016) 16–30.
[57] A.J. Yu and J. Seif, Minimizing tardiness and maintenance costs in flow shop scheduling by a lower-bound-based GA. *Comput. Ind. Eng.* **97** (2016) 26–40.
[58] X.Y. Yu, Y.L. Zhang, D.H. Xu and Y.Q. Yin, Single machine scheduling problem with two synergetic agents and piece-rate maintenance. *Appl. Math. Modell.* **37** (2013) 1390–1399.
[59] F. Zammori, M. Braglia and D. Castellano, Harmony search algorithm for single-machine scheduling problem with planned maintenance. *Comput. Ind. Eng.* **76** (2014) 333–346.
[60] X.G. Zhang, Y.Q. Yin and C.C. Wu, Scheduling with non-decreasing deterioration jobs and variable maintenance activities on a single machine. *Eng. Optim.* **49** (2017) 84–97.