RAIRO-Oper. Res. 56 (2022) 3895–3913
https://doi.org/10.1051/ro/2022191

RAIRO Operations Research
www.rairo-ro.org

A MAKESPAN MINIMIZATION PROBLEM FOR VERSATILE DEVELOPERS IN THE GAME INDUSTRY

Chung-Ho Su[1] and Jen-Ya Wang[2,*]

Abstract. Today, the development of a modern video game draws upon multiple areas of expertise. Moreover, its development cost could be as high as tens of millions of dollars. Consequently, we should carefully schedule its jobs so as not to increase the total cost. However, project leaders traditionally treat developers alike or even schedule all the jobs manually. In this study, we consider a versatile-developer scheduling problem. The objective is to minimize the makespan of a game project. We propose a branch-and-bound algorithm (B&B) to generate the optimal schedules for small problem instances. On the other hand, an imperialist competitive algorithm (ICA) is proposed to obtain approximate schedules for large problem instances. Lastly, computational experiments are conducted to show the performances of both algorithms. When the problem size is small (*e.g.*, $n \leq 12$), B&B can generate the optimal schedules within 5 s. For some large problem instances (*e.g.*, $n = 600$), near-optimal schedules can be obtained by ICA within 10 min. The final results imply that both algorithms converge quickly and are of high solution quality.

Mathematics Subject Classification. 90C05, 90-08.

Received January 12, 2022. Accepted October 25, 2022.

1. Introduction

Multi-machine scheduling is an important research topic in the field of job scheduling. First, from a customer's viewpoint, his/her satisfaction can be improved if these machines are fully utilized. For the same price, the customer usually would prefer an early shipping date. Second, from an enterprise's viewpoint, parallel environments are common in the real world. Multi-machine scheduling is helpful to reduce business costs, such as tardiness penalties. Third, from a researcher's viewpoint, such optimization problems are of great research interest. Most of them are NP-hard, even for the simplest case of $P2||C_{max}$ [49]. Multi-machine scheduling is a generalization of single-machine scheduling, and it is also a special case of flexible flow shops. Hence, any further progress in multi-machine scheduling will benefit the above basic research.

The scheduling of heterogeneous machines is more complicated than that of identical machines. When scheduling jobs on multiple identical machines, we do not need to consider the permutations of the machines. Any one of them can be the leading machine, since they are identical. Even so, minimizing the makespan of two identical machines, *i.e.*, $P2||C_{max}$, is still NP-hard. In a heterogeneous environment, however, the situation is more

Keywords. Job scheduling, game development, makespan, branch-and-bound algorithm, imperialist competitive algorithm.

[1] Department of Animation and Game Design, Shu-Te University, Kaohsiung City, Taiwan, R.O.C.
[2] Department of Multimedia Game Development and Application, Hungkuang University, Taichung City, Taiwan, R.O.C.
*Corresponding author: jywang@sunrise.hk.edu.tw

complicated. Consider two machines with different capabilities, *e.g.*, 10 and 1 job/h. For a two-unit-job project, it is undesirable to have each machine process one unit job. We had better allocate both unit jobs to machine 1, and the makespan will be only 0.2 h. That is, we need to consider all the permutations of jobs on different machines. For example, Alidaee *et al.* [2] aimed to minimize the total tardiness on different machines. These machines were numbered and organized, respectively. Kayvanfar *et al.* [23] also minimized the makespan over unrelated machines. In contrast to a common definition of processing time p_j, they defined a specific symbol p_{ji} for job j assigned to machine i. Both imply that the scheduling is more complicated on unrelated machines than that on homogeneous ones.

Since most heterogeneous machine scheduling problems are NP-hard, metaheuristic algorithms are employed to obtain near-optimal solutions. For example, Khalilpourazari *et al.* [26] proposed a grey wolf optimizer which can quickly converge to a local minimum based on gradient descending. To improve the feasibility, Doulabi *et al.* [12] used only a few parameters to accelerate their convergence speed. On the other hand, Khalilpourazari *et al.* [25] proposed an interesting stochastic fractal search to improve their solution quality; however, their approach required the setting of a dozen parameters. Moreover, a learning-based algorithm was proposed in [24] to predict near-optimal solutions efficiently, *i.e.*, higher execution speed. However, none of the above approaches can ensure solution quality and achieve optimality. For the evaluation of the solution quality of such metaheuristic algorithms, some exact algorithms are needed to disclose the gaps between their approximate solutions and the optimal ones. For more recent metaheuristic algorithms and their solution quality, readers can refer to [11,43,58].

Compared with multi-machine scheduling, multi-developer scheduling is more interesting but also more complicated. In a heterogeneous-machine environment, *e.g.*, [63, 64], a capable machine always processes jobs efficiently in terms of processing speed. However, in the game industry, there are many types of jobs, such as storyboarding, storytelling, prototyping, figure modeling, scene design, sound design, visual effects, rendering, physics, mechanics, programming, and testing [58]. In general, developers with only one specialty cannot easily survive in this industry; they need to equip themselves with several. For example, after finishing the figure modeling, a developer might be asked to perform some testing. Developers are not omnipotent or omniscient, either. Some may excel in figure modeling but be mediocre in programming. Clearly, a versatile developer succeeds in only some specialties. It depends on what types of jobs we assign to him/her. In light of the above observations, scheduling these versatile developers is more complicated than scheduling non-uniform machines. Consequently, new multi-developer scheduling algorithms are called for, rather instead of manual project management.

Three kinds of resources (*i.e.*, finance, manpower, and time) are needed during game development and they should be considered as a whole. First, the cost of developing a large online game is considerably high. For example, the cost of Grand Theft Auto V was at least \$10 000 000 [3, 14, 15, 58]. Second, the team size of such a large game may range from 3 to 100 different developers [40, 58]. Clearly, we could hardly schedule them by our hands alone. Third, the time management of a large game is also a top priority. In general, a developer' annual salary is at least \$66 000 [58]. For some critical jobs, poor time management might lead to heavy penalties. In light of these observations, these resources should be well organized and scheduled in advance. That is, a small makespan could be regarded as an indicator of good resource management.

Today, big data can be used to predict or at least estimate the performance of a developer. For example, Lin *et al.* [36] aimed to minimize the makespan for ordinary manufacturing industries. They employed big data and machine learning to estimate the processing times of jobs. In [34], big data and machine learning were utilized to predict human behaviors in a smart home environment. With these big data, each operation of a single user could be recorded and entered into a database. Therefore, estimating each individual's processing times for different types of jobs will no longer be out of reach. In [28], big data was used to establish a knowledgebase. Referring to failure probabilities, operators could perform various technological processes at the operational level. Moreover, each machine's remaining life could become predictable for a specific operator after a run-in period. The above observations suggest that big data can help us to estimate an operator's processing time if we assign him/her a particular type of job.

Makespan is an important issue in both the manufacturing industry and the game industry. In operations research, the makespan is defined as the total length of a project from beginning to end. In general, a project

leader aims to minimize the makespan to reduce the time cost, resource consumption, and human resources as well as to maximize the profit and customer satisfaction. Such makespan minimization problems are common in many industries, such as the semiconductor industry [60], aviation industry [8], building industry [20], and design industry [29]. In these industries, makespan minimization effectively reduces their costs and increases customer satisfaction. On the other hand, in the game industry, a medium project usually costs a game company at least one million dollars. Furthermore, the cost of a large game like Grand Theft Auto V can be as high as several hundred million dollars [58]. In fact, many game companies run considerable risk and face great financial pressure. For example, Supercell needs to pay Gree 92 million dollars in damages after a mobile-game patent verdict [7]. Intuitively, all of these project leaders need to organize their jobs within a controllable time span. All the above examples show that uncertain or manual project management cannot be used for such large-scale projects. Efficient makespan minimization algorithms for game development are therefore called for.

To our best knowledge, few studies have focused on job scheduling in the game industry, especially for versatile developers. Since some jobs, such as figure modeling, are intangible and inconvenient to quantify in this industry, some project leaders still schedule their jobs manually. Such manual and uncertain scheduling may adversely impact subsequent jobs, such as testing and release. Nowadays, with big data, we can estimate and quantify the processing times of such intangible jobs easily. After setting the scheduling model with proper values, *e.g.*, each job's processing time or a developer's proficiency, makespan minimization in the game industry will become more efficient and effective than it was in the past.

In this study, we aim to minimize the makespan of a project in the game industry. As discussed earlier, we cannot equate a versatile developer with a heterogeneous machine, unless all the jobs in the game industry degenerate into only a single type. This restriction implies that the problem is more complicated and some new algorithms are needed. First, we propose an exact algorithm, *i.e.*, a branch-and-bound algorithm (B&B), to generate the optimal schedules as a benchmark for evaluating other algorithms' solution qualities. Second, an approximate algorithms, *i.e.*, an imperialist competitive algorithm (ICA), is also developed for providing approximate schedules. The reasons are that the problem size, *i.e.*, number of jobs, is usually larger than 100 in the real world and no exact algorithm can provide real-time solutions to this NP-hard problem.

The rest of this paper is organized as follows. Section 2 introduces some related studies. In Section 3, a makespan minimization problem in the game industry is presented. In Section 4, we develop a branch-and-bound algorithm to generate the optimal schedules when the problem size is small. In Section 5, an imperialist competitive algorithm is proposed to deal with larger problem instances. Some computational experiments are conducted to evaluate the two algorithms in Section 6. Finally, conclusions are presented in Section 7.

2. Related work

In this section, some exact and approximate algorithms are introduced and discussed. Since they still have some shortcomings, these existing algorithms cannot be directly applied to the presented problem.

2.1. Branch-and-bound algorithms

Branch-and-bound algorithms are a popular solution technique for obtaining exact solutions in the field of job scheduling. Table 1 divides these branch-and-bound algorithms into two types: unifunctional machines and multifunctional ones. For example, in this study, a versatile developer excels in programming but is mediocre in scene modeling; *i.e.*, the developer is a kind of multifunctional machine. For unifunctional machines, a capable one always processes jobs at a steady speed; however, a developer's processing speeds may fluctuate between 1 and 10 jobs/day, depending on the job types. That is, one should not be simply rated as a versatile developer due to his/her high processing speed for a single type of jobs only. Consequently, these branch-and-bound algorithms cannot be directly applied to the presented problem.

Developing an exact algorithm for this problem is of great importance. Clearly, branch-and-bound algorithms are of little practicality for the real-world problem instances. For example, branch-and-bound algorithms schedule identical machines well only for small instances, *e.g.*, $n \leq 15$ in [63]. However, the optimal solutions can be used

TABLE 1. Some existing branch-and-bound algorithms for job scheduling.

Type	Unifunctional machine(s)			Multifunctional machine(s)/Versatile developer(s)	
Objective	Single	Identical	Heterogeneous	Unrelated	Versatile
Tardiness	[27, 66, 70]	[30, 42, 61]	[64]	[5, 23, 41]	
Completion time	[22, 44, 69]	[17, 33, 45]		[48]	
Makespan	[62, 68]	[46]	[54]	[18]	[This study]

TABLE 2. Some existing metaheuristic algorithms for job scheduling.

Type	Unifunctional machine(s)			Multifunctional machine(s)/developer(s)	
Objective	Single	Identical	Heterogeneous	Unrelated	Versatile
Tardiness	[13, 39, 71]	[38, 52, 53]	[16]	[35, 41, 57]	
Completion time	[51, 59]	[10]		[50, 72]	
Makespan	[1, 47]	[55]	[56]	[4, 21, 37]	[This study]

as benchmarks for evaluating other metaheuristic algorithms. Without these exact algorithms, comparing two metaheuristic algorithms (*e.g.*, GA and ACO) seems meaningless. Maybe both of them easily become trapped in some local minimums. Therefore, thousands of studies on branch and bound algorithms are proposed each year to obtain the optimal solutions.

2.2. Metaheuristic algorithms

Metaheuristic algorithms are helpful for generating near-optimal solutions when problem sizes are large. Table 2 lists some metaheuristic algorithms for scheduling machines in the field of job scheduling. Again, those approximate algorithms designed for unifunctional machine(s) are not suitable for the presented problem because the processing speed of each machine is always fixed, *i.e.*, inflexible, in our problem. On the other hand, although some approximate algorithms have been proposed for unrelated machines, they are too time-consuming because there are $m \times n$ relationships (*e.g.*, p_{ij}) among m different machines and n various jobs, *i.e.*, a very large solution space. In the real world, in fact, there are only several job types; *i.e.*, each single job rarely forms a job type. To our best knowledge, no past research has studied such job scheduling problems in the game industry. Some efficient approximate algorithms are required.

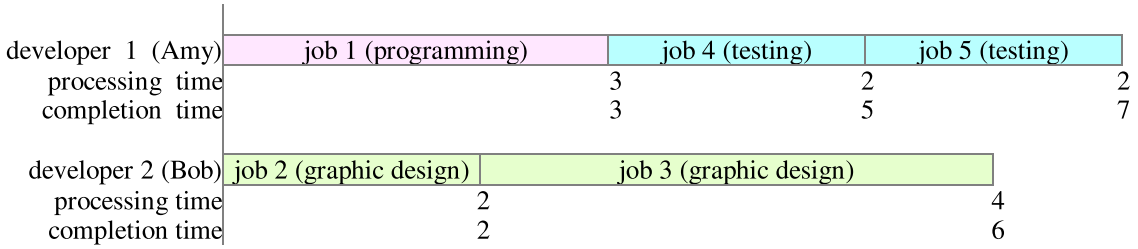
3. PROBLEM DEFINITION

The scheduling problem is defined as follows. There are m developers about to undertake a game project consisting of n jobs. Each job j has a default processing time $p_j \in \mathbf{Z}^+$ and a job type $e_j \in \{1, 2, 3\}$ and needs to be assigned to one developer only. Each developer processes one job at a time. Let $r_{xe} \in [0, 1)$ denote the proficiency of developer x processing a job of type e. Namely, if job j is assigned to developer x and $e_j = a$, the actual processing time is $p_j(1 - r_{xe_j}) = p_j(1 - r_{xa})$. For a schedule π, the completion time of jobs j is denoted by $C_j(\pi)$. Under the above assumptions, the problem is to minimize the makespan of the game project. That is, the objective function is defined as

$$\text{Minimize } f(\pi) = \max_{x=1}^{m}\{\max\{C_j(\pi)|\text{ job } j \text{ is assigned to developer } x\}\}.$$

	job type 1 (programming)	job type 2 (graphic design)	job type 3 (testing)
developer 1 (Amy)	$r_{11} = 0.5$	$r_{12} = 0.4$	$r_{13} = 0.5$
developer 2 (Bob)	$r_{21} = 0.2$	$r_{22} = 0.5$	$r_{23} = 0.1$

(a)



(b)

FIGURE 1. A problem instance. (a) The proficiency ratios of two developers. (b) A schedule π and its objective cost.

TABLE 3. The notations and meanings.

Symbol	Meaning
m	The number of developers
n	The number of jobs
p_j	The processing time of job j, where suffix j means job Id
$e_j = a$	The type of job j, where suffix $j \in \{1, 2, \ldots, n\}$ means a job Id and $a \in \{1, 2, 3\}$ means a job type
r_{xe}	The proficiency of developer x processing a job of type e, suffix $x \in \{1, 2, \ldots, m\}$ means a developer and suffix $e \in \{1, 2, 3\}$ means a job type
π	A schedule or a permutation of all the n jobs (*i.e.*, a decision variable)
$\pi = (\alpha, \beta)$	A partially determined schedule, where α means a determined partial schedule and β means a the set of the remaining undetermined job Ids
$C_j(\pi)$	The completion time of job j for a given schedule π

A problem instance is shown in Figure 1. Consider that there are two developers ($m = 2$) and five jobs ($n = 5$) with $e_j = 1, 2, 2, 3, 3$, $p_j = 6, 4, 8, 4, 4$, for $j = 1, 2, 3, 4, 5$. Clearly, developer Amy is proficient at programming and testing, and developer Bob is good at graphic design. Consequently, we assign jobs 1, 4, and 5 to Amy and jobs 2 and 3 to Bob. That is, we have a schedule $\pi = (1, 4, 5, 0, 2, 3)$, where the zero represents a separator that divides jobs between these developers. For example, the actual processing time of job 1 is $3 (= 6 \times (1 - 0.5))$, since job 1 is assigned to developer 1, $e_1 = 1$, and $r_{11} = 0.5$. For this problem instance, each job is assigned to its best-fit developer. Therefore, the makespan is 7, *i.e.*, the completion time of the last job.

For convenience, all the related symbols used in the problem are listed in Table 3. Note that m, n, p_j, e_j, and r_{xe}, are all given constants, and π is the decision variable of this presented problem.

The following lemma shows that each problem instance has a bound for its objective cost. No matter how we schedule these jobs, the optimal cost is never lower than this bound. Later, this property can help us to develop an efficient branch-and-bound algorithm.

Lemma 1. *For each job j, let $r_j^{\max} = \max_{x=1}^{m}\{r_{xe_j}\}$. The makespan of the presented problem is at least $\sum_{j=1}^{n} p_j(1-r_j^{\max})/m$.*

Proof. This problem can be proved by contradiction. Suppose that there exists an optimal schedule π^* and it generates a makespan $f(\pi^*) < \sum_{j=1}^{n} p_j(1-r_j^{\max})/m$. That is, for each developer x, we have $\max_{x=1}^{m}\{C_j(\pi)|j@x\} < \sum_{j=1}^{n} p_j(1-r_j^{\max})/m$ for all $x = 1, 2, \ldots, m$. On the other hand, we have

$$\begin{aligned} m \times f(\pi^*) &= m \times \max_{x=1}^{m}\{C_j(\pi)|j@x\} \\ &\geq \sum_{x=1}^{m}\left[\sum_{j@x} p_j\left(1-r_{xe_j}\right)\right] \quad \text{(Not all developers take the makespan } f(\pi^*)\text{)} \\ &\geq \sum_{x=1}^{m}\left[\sum_{j@x} p_j\left(1-r_j^{\max}\right)\right] \quad \text{(Let each job be processed by its best-fit developer)} \\ &= \sum_{j=1}^{n} p_j\left(1-r_j^{\max}\right). \end{aligned}$$

Then we obtain $f(\pi^*) \geq \sum_{j=1}^{n} p_j(1-r_j^{\max})/m$. It contradicts the initial supposition. The proof is complete. □

The presented problem is NP-hard. Even for a simple case named X, *i.e.*, $r_{ij} = 0$ for all i and j, the presented problem is still NP-hard. Before proving this property, a determined NP-hard problem is introduced for later problem reduction. Consider a well-known NP-hard problem, *i.e.*, sub-set sum [9], as follows. Given a set of n positive integers, *i.e.*, $R = \{r_1, r_2, \ldots, r_n\}$, and a target t, the problem named Y is to check if there exists a subset whose elements sum up to t. The following lemma will prove the NP-hardness by reducing problem Y to problem X.

Lemma 2. *Problem X is NP-hard.*

Proof. First, given an instance of problem X, we build a corresponding instance of problem Y. Then, let $P = \sum_{j=1}^{n} p_j$. Next, we construct a corresponding instance of Y by letting $r_1, r_2, \ldots, r_n$ be n m-digit $(P+1)$-ary numbers and r_{ji} denoting the i-th significant digit of number r_j. Finally, let

$$r_{ji} = \begin{cases} p_j, & \text{if job } j \text{ is assigned to developer } i, \\ 0, & \text{otherwise,} \end{cases}$$

for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

Now we show that problem Y has a solution if and only if problem X has a solution. Let t be an m-digit $(P+1)$-ary target with the largest digit t_k for problem Y. Then, given a solution of Y, *i.e.*, a subset R', we need to show that there exists a schedule π' such that $f(\pi') = t_k$ for problem X. According to t, let $\sum_{\text{job } j \text{ is assigned to developer } i}(p_j) = t_i$ for $i = 1, 2, \ldots, m$, where t_i is the i-th significant digit of t. Therefore, the solution of problem X, *i.e.*, $f(\pi') = t_k$, is found.

Conversely, assume that π'' is the solution of problem X. According to π'', let $t_i = \sum_{\text{job } j \text{ is assigned to developer } i}(p_j)$ for $i = 1, 2, \ldots, m$, where t_i is the i-th significant digit of the m-digit $(P+1)$-ary target t. Note that there exists $t_k = f(\pi'')$ for some k. Then, for problem Y, there exists a subset R'', *i.e.*, R itself, in which the summation of its elements is t. Note that problem Y is NP-hard. Hence, problem X is also NP-hard. The proof is complete. □

Clearly, the presented problem is different from those in past research. Consequently, some new scheduling algorithms are called for. The following observations can help us to develop more efficient scheduling algorithms.

Observation 1. Intrinsically, this presented problem is a partition problem. By dividing these jobs, the optimality, *i.e.*, minimum makespan, can be achieved by partition instead of permutation.

Observation 2. Load balance is emphasized in traditional multi-machine scheduling. However, for the presented problem, there might be an evenness anomaly. For example, some developers capable of handling jobs of type 1, they might be idle in an optimal schedule. That is, the numbers of jobs assigned to each developer are not even. As it happens, in this problem instance, no type-1 jobs need to be processed.

Observation 3. For tardiness minimization problems, the minimum costs can be zero. However, for the presented problem, the minimum makespan will not be shorter than $\sum_{j=1}^{n} p_j(1 - r_j^{\max})/m$.

Observation 4. For completion time minimization problems, the goal is to minimize the average completion time of all jobs. However, for the presented problem, we aim to reduce the completion time of the last job only. That is, the optimal schedules of the former problems are not necessarily equal to that of the latter problem.

Observation 5. For traditional heterogeneous-machine scheduling problems, given the same job, the most efficient machine always outperforms other machines in terms of throughput. These efficient machines dominate the game. However, in the presented problem, there is no pre-decided winner. The success depends on the complementarity of all the developers.

4. Branch-and-Bound algorithm

In this section, we propose a branch-and-bound algorithm (named B&B) for generating the optimal schedules. First, some dominance rules are developed. Then, a lower bound is proposed for accelerating B&B's speed. Lastly, B&B traverses a search tree in the depth-first-search (DFS) order. This exact algorithm can help us to measure solution quality; *i.e.*, it can serve as a benchmark for evaluating other approximate algorithms.

4.1. Dominance rules

For convenience, we introduce some simple notations to develop the following rules. Suppose that we have an incomplete schedule $\pi = (\alpha, \beta)$, where α is a determined partial sequence and β is not. Since B&B proceeds in the DFS order, some root-to-leaf schedules have been visited halfway. Therefore, we can keep track of the currently minimal cost, *i.e.*, C^*, at any time. Moreover, we assume that the ID of each job j is number j and the last job of α is assigned to some developer x.

In Rule 1, each of the two developers is assigned a job he/she dislikes. If we interchange the two jobs and the original objective cost can be reduced, this rule holds. Since these proofs of these dominance rules are similar, due to the limited length, we provide only the first one.

Rule 1. Let jobs j and i be the last two jobs in α with $e_j = b$ and $e_i = a$, job j be assigned to developer y, and job i be assigned to developer x. If $p_j(1 - r_{yb}) > p_i(1 - r_{ya})$ and $p_i(1 - r_{xa}) > p_j(1 - r_{xb})$, then π is dominated.

Proof. Let us interchange the two jobs and observe their outcomes. The original processing times of the two jobs are $p_j(1 - r_{yb})$ and $p_i(1 - r_{xa})$, respectively. After interchanging, the resultant processing times of the two jobs are $p_i(1 - r_{ya})$ and $p_j(1 - r_{xb})$, respectively. Clearly, both developers' processing times decrease. That is, the interchange will not lead to any makespan gain. The proof is complete. □

Rule 2 shows that a schedule of a developer cannot end too late, *i.e.*, he/she will be overloaded. In contrast, Rule 3 shows that we cannot let a capable developer be idle too early.

Rule 2. Let i be the last job of α. If $C_i(\alpha) > C^*$, then π is dominated.

Rule 3. Let i be the last job of α and assigned to developer x. If there exists a job k of some type a in β such that $C_{\text{job } i \text{ is assigned to developer } x}(\alpha) + p_{k \in \beta}(1 - r_{xa}) \leq C^*$, then π is dominated.

Without loss of generality, we assume that all the jobs assigned to a single developer are sorted in ascending order. This assumption does no harm to the optimality. Rule 4 shows that the ID's of two adjacent jobs must be increasing. Similarly, if developer x is the last available developer, his/her leading job ID must be the minimal one in the remaining jobs.

Rule 4. Let j and i be the last two jobs in α assigned to a developer. If $j > i$, then π is dominated.

Rule 5. Let developer x be the last developer and his/her leading job is job i. If there exists a job $k < i$ in β, then π is dominated.

Let developer x be the last developer. Clearly, we need to assign all the remaining jobs of β to him/her. If his/her maximum completion time is larger than the current one, then Rule 6 will trim this branch.

Rule 6. Let developer x be the last developer and the last job i in α be processed by him/her. If $C_{\text{job } i \text{ is assigned to developer } x}(\alpha) + \sum_{k\in\beta}(1 - r_{xe_k}) > C^*$, then π is dominated.

4.2. Lower bound

When searching a search tree, B&B supposedly needs to browse all of the root-to-leaf paths in the DFS order. However, sometimes, some paths are not worth following to the end. For example, at the beginning, no jobs are assigned to developer 1, despite the availability of many jobs suitable for him/her. Such a root-to-leaf path (*i.e.*, a schedule) can be eliminated from a search tree as early as possible, since this path will never be an optimal solution. Consequently, a lower bound, named LB, is developed to eliminate these useless paths.

Figure 2 depicts a lower bound to accelerate the exact algorithm, *i.e.*, B&B. Let C^* be the currently minimal cost that B&B has ever recorded so far, since B&B is still only halfway to searching all the root-to-leaf paths. For a root-to-leaf path $\pi = (\alpha, \beta)$, we can assume the leading nodes, *i.e.*, α, have been visited and the remaining jobs have not been visited. Since B&B proceeds in the DFS order, we can assume that developers a, a+1, ..., m are available and that their individual makespans are set in Steps 1 and 2. Then the remaining jobs in β are grouped by their job types and the total workloads of the three job types are accumulated in Steps 3 and 4. In Step 5, a maximal proficiency $r_{m^*e^*}$ is determined; *i.e.*, some available developer m^* is of the maximal proficiency $r_{m^*e^*}$ for some remaining job type e^*. If developer m^* still has a lot of capacity to accept jobs, let him/her finish all the jobs of type e^*; *i.e.*, no jobs of type e^* remain. Moreover, the makespan of developer m^* needs to be reset (Steps 9 and 10). On the other hand, if the developer will be fully-loaded soon, we allocate to him/her only some of jobs of type e^*, which keep him/her busy until C^*. Note that he/she is no longer able to accept jobs. Since jobs of type e^* still remain, we reset the remaining workload in Steps 12 and 13. Thus far, there might be some fully-loaded developers or some types of jobs are all finished. Consequently, we need to determine another new developer m^* who excels in another type of remaining jobs (*i.e.*, e^*) and has a new maximal proficiency $r_{m^*e^*}$ in the end of the for loop (Step 14). Repeating the for loop, if all the workloads are digested, the estimated lower bound is returned in the last step.

4.3. Main program

Figure 3 shows our proposed branch-and-bound algorithm, *i.e.*, B&B. Supposedly, B&B needs to visit every root-to-leaf path in a search tree one by one in the DFS order. Due to the above dominance rules and lower bound, B&B can prune some useless branches in advance and accelerate its searching speed. Suppose that B&B is searching a search tree halfway; *i.e.*, only some jobs in a root-to-leaf path are determined. Let the determined partial sequence be α, the number of jobs in α be d, developer a process the last job of α, and the remaining jobs be β (Steps 1 and 2). Since B&B is designed in the DFS order, it implies that the workloads of developers $1, 2, \ldots, a-1$ are all determined by α. That is, so far, developer a is semi-loaded, and the workloads of the remaining developers are all undetermined. Since developers $1, 2, \ldots$, and a are assigned more or fewer jobs, the currently heaviest workload of a developer (*i.e.*, his/her makespan) is known and denoted by $t^{\max} = f(\alpha)$ in Step 3. If the schedule $\pi = (\alpha, \beta)$ is dominated by a rule or its estimated makespan is larger than the current

Algorithm $LB(\alpha, \beta, C^*)$
INPUT
α: the determined partial sequence in a schedule $\pi = (\alpha, \beta)$
β: the undetermined part
C^*: the currently minimal makespan that B&B has obtained so far
OUTPUT
C^{lb}: the lower bound for the specific schedule $\pi = (\alpha, \beta)$

---

1) Let a be the ID of the developer processing the last job of α and the completion time be t_a;
2) **For** $i = a+1$ **to** m do $t_i = 0$; //set the initial makespan for each reaming developer
3) **For** $e = 1$ **to** 3 do $w[e] = 0$;
4) **For** job j **in** β do $w[e_j] = w[e_j] + p_j$;//accumulate the workloads of three job types
5) Let $r_{m^*e^*} = \max\{r_{ij} \mid \text{for any available developer and any remaing job type}\}$;
6) Set $sum = t_a$; //start to accumulate the total processing time
7) **Repeat** Steps 8–15 **until** all the workloads are zeros, i.e., $w[e] = 0$ for all e = 1, 2, 3;
8) **If** $(t_{m^*} + w[e^*](1 - r_{m^*e^*}) \le C^*)$ **then** do Steps 9–10;
9) Set $sum = sum + w[e^*](1 - r_{m^*e^*})$;
10) Let $t_{m^*} = t_{m^*} + w[e^*](1 - r_{m^*e^*})$ and $w[e^*] = 0$;
11) **Else** do Steps 12–13;
12) Set $\Delta = C^* - t_{m^*}$ and $sum = sum + \Delta$;
13) Let $t_{m^*} = C^*$, $w[e^*] = w[e^*] - \Delta / (1 - r_{m^*e^*})$, and developer m^* be unavailable;
14) Let $r_{m^*e^*} = \max\{r_{ij} \mid \text{for any available developer and any remaing job type}\}$;
15) Set $C^{lb} = sum / (m - a + 1)$ and return C^{lb}.

FIGURE 2. The pseudo code of the lower bound.

one, the schedule will be pruned in Steps 4 and 5 to avoid further meaningless searches. If B&B is at a leaf node (*i.e.*, the schedule is determined) and the makespan is shorter than the current C^*, then C^* will be replaced by the lower one (Steps 6 and 7). If B&B is at a middle node, all possible permutations of β are fabricated and we let B&B explore each permutation one by one in the DFS order. As all the root-to-leaf paths are either visited or pruned, the optimal schedule and the minimal makespan are stored in the two global variables, *i.e.*, π^* and C^*, respectively.

With the above exact algorithm, we can obtain the optimal schedules for some small problem instances (*e.g.*, n = 12). Although B&B cannot be applied to some large instances in the real world, it can be used as a benchmark to evaluate some metaheuristic algorithm's performance, *e.g.*, GA or ACO. Consequently, it is still of great research interest.

Algorithm $B\&B(\alpha,\beta)$
INPUT
α: the determined partial sequence in a schedule $\pi=(\alpha,\beta)$
β: the undetermined part
OUTPUT
π^*: the optimal schedule //global variable
C^*: the minimal makespan that B&B has recorded so far //global variable
---
1) Let a be the ID of the developer processing the last job of α;
2) Let d be the depth of the DFS search, i.e., the number of jobs in α;
3) Let $t^{\max}=f(\alpha)$ be the maximum of the first a developers' makespans;
4) **If** π is not dominated by rules 1–6 **then** do Steps 5–14;
5) **If** $(\max\{t^{\max}, LB(\alpha,\beta,C^*)\}\le C^*)$ **then** do Steps 6–14;
6) **If** $(d=n)$ **then** do Step 7; //at a leaf node
7) **If** $(f(\pi)<C^*)$ **then** let $\pi^*=\pi$ and $C^*=f(\pi)$;
8) **Else** do Steps 9–14; //at a middle node of the search path, i.e., π
9) **For** $j=d+1$ **to** n do Steps 10–14;
10) Let $\pi'=\pi$; //Get a new copy of π
11) Swap the two jobs of π' at positions $d+1$ and j;
12) Let α' be the partial sequence of the first $(d+1)$ scheduled jobs in π';
13) Let β' be the remaining jobs;
14) Execute $B\&B(\alpha',\beta')$ recursively.

FIGURE 3. The pseudo code of the branch-and-bound algorithm.

5. IMPERIALIST COMPETITIVE ALGORITHM

A metaheuristic algorithm is developed in this section to provide approximate solutions to large problem instances in the real world. Compared with an ordinary genetic algorithm, an imperialist competitive algorithm has a better capability for diversification; *i.e.*, it can explore a wider space [6]. The reason is that a genetic algorithm, in general, evolves within a single population, whereas an imperialist competitive algorithm evolves within several empires. The emergence of a locally optimal solution will not cause a premature convergence of all the empires. That is, an imperialist competitive algorithm constructs its fire wall to avoid some adverse bandwagon effects [65].

Figure 4 presents the proposed imperialist competitive algorithm (named ICA). At the beginning, N citizens are randomly generated in Step 1, where a citizen is a random permutation of the ID's of n jobs. Consider the example shown in Figure 1 again. Citizens 1, 4, and 5 are assigned to empire 1, and citizens 2 and 3 are assigned to empire 2. Then a schedule π is encoded as (1, 4, 5, 0, 2, 3), where the number 0 means a separator. In Steps 2 and 3, each citizen is evaluated and randomly assigned to an empire. If a citizen has a shorter makespan, *i.e.*, a lower objective cost $f(\pi_i)$, he/she has a greater chance to survive into a next generation, *i.e.*, $[1/f(\pi_i)]/[\sum_{k=1}^{N} 1/f(\pi_k)]$. So far, ICA can determines each empire's ruler, defined as the best citizen having the lowest objective cost in his/her empire. Step 4 sets the initial values for parameters G, T, π^+, and C^+, where G means the current generation, T the elapsed execution time, π^+ the currently optimal schedule, and

TABLE 4. The default values of the parameters.

Parameter	Default value	Range	Meaning
m	3	$2, 3, 5$	The number of developers
n	12	$8, 10, 12, 200, 400, 600$	The number of jobs
D^D	0	$0, 1, 2, 3$	The distribution of developers
D^J	3	$0, 1, 2, 3$	The distribution of jobs
p_j		$1, 2, \ldots, 100$	The processing time of job j
r_a	0.90	$0.50, 0.55, \ldots, 0.95$	The assimilation rate of ICA
r_r	0.10	$0.05, 0.10, \ldots, 0.50$	The rebellion rate of ICA
r_{ls}	0.02	$0.00, 0.01, \ldots, 0.05$	The local search rate of ICA
M	3	3	The number of empires
N	500	500	The population size
n	12	$8, 10, 12, 200, 400, 600$	A stopping criterion, *i.e.*, the number of jobs
G	500	500	A stopping criterion, *i.e.*, the number of unimproved generations

C^+ the globally minimal objective cost. Then, a standard roulette wheel selection [67] is employed to select r_aN citizens and they will be forced to randomly move towards their corresponding rulers, respectively. Step 6 adopts a crossover operation, PMX crossover [19]. Next, r_rN citizens are randomly chosen for adjustment in Step 7; *i.e.*, either some jobs within a citizen are randomly swapped or the citizen is forced to betray his/her empire. The adjustment is implemented by a shift-and-insert mutation [32]. In Step 8, only a few citizens are chosen for modification again; *i.e.*, a local search is performed [67]. At the end of each generation, Steps 9–13 reassign a citizen of the weakest empire to another empire, wherein the weakest empire is the one with the ruler having the largest objective cost among the M rulers. Moreover, if some newly-born citizen has the lowest objective cost, we reset the stopping criterion $G = 0$ and let him/her be the new ruler of his/her empire and record the current information. If execution time T is less than n seconds and the currently optimal schedule is frequently updated, let Steps 6–13 repeat again. Otherwise, we end this algorithm and return the near-optimal schedule π^+.

6. EXPERIMENTAL RESULTS

In this section, we first examine the execution speed of B&B. Then we evaluate the solution quality and execution speed of ICA. Moreover, ICA is also compared with a typical genetic algorithm (named GA) [53]. Lastly, we perform a sensitivity test for observing the influence of versatile developers on the objective cost.

Table 4 lists the default values of the parameters. As defined in Section 2, m and n are the numbers of developers and jobs, respectively. We design four distributions of developers for $D^D = 1$, 2, 3, and 4. First, for $D^D = 0$, let all the developers be mediocre, *i.e.*, $r_{xe} \in [0.1, 0.4]$. Second, for $D^D = 1$, let all the developers be uni-specialty experts. That is, a developer x has $r_{xe} \in [0.6, 0.9]$ for only some single type of $e \in \{1, 2, 3\}$ and $r_{xe} \in [0.1, 0.4]$ for the other two types of jobs. Third, for $D^D = 2$, all the developers are bi-specialty experts. That is, a developer x has $r_{xe} \in [0.6, 0.9]$ for two random job types and $r_{xe} \in [0.1, 0.4]$ for the other remaining type. Fourth, for $D^D = 3$, all the developers are versatile experts, *i.e.*, $r_{xe} \in [0.6, 0.9]$ for all x and e. On the other hand, note that there are three types of jobs. We also design four different distributions of jobs: D^J for $J = 1$, 2, 3, and 4. First, for $D^J = 0$, all the jobs are of an identical processing time and belong to only some single type. Second, for $D^J = 1$, all the jobs are of an identical processing time and belong to two types at random. Third, for $D^J = 2$, all the jobs randomly belong to two types and the processing time of each job is randomly chosen from $\{1, 2, \ldots, 100\}$. Fourth, for $D^J = 3$, all the jobs randomly belong to three types and the processing time of each job is randomly chosen from $\{1, 2, \ldots, 100\}$. Lastly, the parameters used in the following

Algorithm *ICA*(r_a , r_r , r_{ls} , M , N , n)
INPUT
r_a : the assimilation rate
r_r : the rebellion rate
r_{ls} : the local search rate
M : the number of empires
N : the number of citizens, i.e., population size
n : the number of jobs, i.e., stopping criterion
OUTPUT
π^+ : the best ruler, i.e., a near-optimal schedule

//Initialization
1) **For** i = 1 **to** N do randomly generate citizen i and assign it to an empire; //i.e., schedule π_i
2) **For** i = 1 **to** N do evaluate the objective cost of citizen i;
3) **For** i = 1 **to** M do determine the ruler of empire i each;//i.e., has the minimal objective cost
4) Set $G = 0$, $T = 0$, $\pi^+ = \pi_1$, and $C^+ = f(\pi_1)$;
5) **While** ($G \le 500$ and $T \le n$) **do** Steps 6–13;
//Assimilation
6) **For** i = 1 **to** $r_a N$ do force a citizen to move towards its ruler at random;
//Rebellion
7) **For** i = 1 **to** $r_r N$ do make a citizen stroll or reassign it to another empire at random;
//Local search
8) **For** i = 1 **to** $r_{ls} N$ do let a citizen stroll around its neighborhood at random;
//Competition
9) Randomly let a citizen of the weakest empire betray its empire;
10) **For** i = 1 **to** N do Steps 11–12;
//Evaluation
11) Compute the objective cost of citizen i, i.e., $f(\pi_i)$;
12) **If** $f(\pi_i) < C^+$ **then** set $G = 0$, $\pi^+ = \pi_i$, and $C^+ = f(\pi_i)$;
13) Record elapsed time T and set $G = G + 1$;
14) Output the best ruler π^+.

FIGURE 4. The pseudo code of the imperialist competitive algorithm.

experiments, *i.e.*, r_a, r_r, r_{ls}, M, N, n, G, are the same as those introduced in the previous sections. A pilot experiment suggests that $r_a = 0.9$, $r_r = 0.1$, and $r_{ls} = 0.02$ lead to higher solution quality and less execution time. All the proposed algorithms are implemented in Pascal and executed on an Intel Core i7@3.20 GHz with 32 GB RAM in a Windows 10 environment. For each setting, 50 random trials are conducted and their statistics are recorded.

TABLE 5. The performance of B&B for $n = 8$, $D^D = 0$, and $D^J = 3$.

	B&B				ICA				GA			
	Nodes		Run Time		Run Time		REP		Run Time		REP	
m	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
2	1419.640	6513	0.004	0.016	3.244	3.531	0.000	0.000	0.144	0.172	0.000	0.000
3	477.980	5809	0.001	0.016	3.470	3.828	0.000	0.000	0.151	0.157	0.400	4.422
5	2079.720	24 234	0.005	0.063	3.662	4.766	0.000	0.000	0.163	0.172	1.713	11.681

TABLE 6. The performance of B&B for $n = 10$ and $D^J = 3$.

		B&B				ICA				GA			
		Nodes		Run Time		Run Time		REP		Run Time		REP	
m	D^D	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
2	0	26 508.800	91 433	0.100	0.313	3.615	3.906	0.000	0.000	0.155	0.172	0.032	0.731
	1	59 514.840	286 223	0.204	0.844	3.623	3.906	0.000	0.000	0.154	0.157	0.034	0.721
	2	105 649.760	1 349 301	0.300	2.500	3.625	4.015	0.000	0.000	0.155	0.172	0.012	0.558
	3	49 598.500	650 949	0.168	1.875	3.709	5.110	0.000	0.000	0.153	0.157	0.008	0.419
3	0	17 386.180	272 369	0.055	0.844	3.880	4.250	0.000	0.000	0.165	0.281	1.240	5.372
	1	12 107.780	322 349	0.039	0.891	3.827	4.125	0.000	0.000	0.161	0.187	0.639	3.965
	2	69 288.560	1 621 025	0.185	4.062	3.785	4.188	0.000	0.000	0.160	0.172	0.953	5.247
	3	5983.100	122 318	0.021	0.344	3.880	5.094	0.000	0.000	0.169	0.250	0.549	3.655
5	0	23 148.080	631 688	0.072	1.938	4.202	5.109	0.000	0.000	0.181	0.265	3.904	11.979
	1	195 300.920	5 216 072	0.524	13.563	3.992	4.875	0.000	0.000	0.177	0.219	2.106	12.770
	2	1 237 123.880	18 961 243	3.244	48.110	3.879	4.922	0.000	0.000	0.175	0.188	0.535	15.718
	3	102113.180	4 194 394	0.275	11.031	4.229	4.703	0.000	0.000	0.180	0.250	2.675	16.689

Table 5 lists the statistics of the three algorithms for $n = 8$. Assume that all the m developers are mediocre and all the n jobs each randomly belong to one of three types. Note that the increase in developers does not necessarily lead to a decrease in nodes for B&B. The reason is that the scale of a search tree increases, so we need to perform more trial and error to locate the optimal schedule. To evaluate the solution quality of both metaheuristic algorithms, we define the relative error percentage (REP) as $(f^{\mathrm{ICA}} - f^*)/f^* \times 100\%$ for ICA, where f^{ICA} is the objective cost of ICA and f^* is the optimal cost obtained by B&B. Similarly, the REP for GA is $(f^{\mathrm{GA}} - f^*)/f^* \times 100\%$. Observing the two REP columns, we learn that ICA always outperforms GA in terms of solution quality. It is clear that ICA requires more run time, which is indeed a shortcoming. However, from the viewpoint of diversification [6], ICA can avoid premature convergence and prevent itself from being trapped in some local minimums. This implies that ICA has a better ability to explore the search space at a cost of run time.

In Table 6, there are 10 jobs, each belonging to a random type. Again, the results show that more developers means more execution time for B&B. In addition, the more consistent the developers are, the less run time B&B takes. That is, a mix of mediocre and versatile developers will cause B&B to require more run time. Note that each system setting requires 50 trials of B&B, 50 trials of ICA, and 50 trials of GA. In a worst case, B&B will take 48.11 s for a single trial. This duration implies that B&B is too time-consuming for large problem instances. On the other hand, ICA remains good solution quality for $n = 10$. The consistency of the developers does not influence the run times of the two metaheuristic algorithms. However, it is more difficult for GA to locate the optimal solutions (*i.e.*, larger REP's), if all the developers are mediocre. In general, the more developers we have, the more run time both approximate algorithms will take.

TABLE 7. The performance of B&B for $n = 12$, $m = 3$, and $D^D = 2$.

	B&B					ICA					GA				
	Nodes		Run Time			Run Time		REP			Run Time		REP		
D^J	Mean	Max	Mean	Max	NA	Mean	Max	Mean	Max	NA	Mean	Max	Mean	Max	NA
0	10 941 247.268	29 500 825	41.061	112.359	9	2.490	2.594	0.000	0.000	9	0.173	0.188	0.000	0.000	9
1	3 971 253.760	71 928 145	13.439	219.594	0	3.310	3.797	0.000	0.000	0	0.169	0.172	0.042	2.101	0
2	1 008 421.480	15 746 313	3.398	47.891	0	4.253	5.469	0.000	0.000	0	0.172	0.234	1.086	5.547	0
3	962 066.080	11 601 099	2.997	30.329	0	4.208	5.235	0.000	0.000	0	0.173	0.219	1.226	5.676	0

TABLE 8. The performance of ICA for $n = 200$, $D^D = 0$, and $D^J = 3$.

	ICA				GA			
	Run Time		RDP		Run Time		RDP	
m	Mean	Max	Mean	Max	Mean	Max	Mean	Max
2	123.523	193.734	0.000	0.000	1.455	4.359	0.872	1.726
3	169.288	200.547	0.000	0.000	1.714	3.953	4.271	8.984
5	182.740	200.765	0.000	0.000	1.698	4.297	7.187	13.918

In Table 7, B&B aims to solve the problem instances of $n = 12$. There are three developers and all of them are very good at processing two types of jobs at random, *i.e.*, $r_{x\varepsilon} \in [0.6, 0.9]$. Unlike the previous two tables, the columns of NA means that the optimal solutions are not available within one billion nodes of B&B. In the first setting, *i.e.*, $D^J = 0$, nine problem instances cannot be optimally solved within 1 billion nodes. That is, $n = 12$ is the maximal problem size that B&B can accept. It is interesting that the variety of jobs can help B&B to converge quickly; this is true because a job of a particular type is likely to be assigned to a corresponding developer at a very low cost. The lower bound can efficiently prune such a subtree. On the other hand, ICA can locate the optimal solutions within 5.5 s. However, jobs of various types will increase the run time of both metaheuristic algorithms. That is, the local minimums of $D^J = 3$ are similar and neither metaheuristic algorithms can tell which is the globally minimal or converge to it.

Table 8 presents the experimental results when the problem size is large, *i.e.*, $n = 200$. The developers are all mediocre and the types of jobs are randomly distributed in 1, 2, or 3. Since B&B cannot play the role of a benchmark for such large problem instances, we define a new measurement. The relative deviation percentage (RDP) for ICA is defined as $(f^{\text{ICA}} - f^{\#})/f^{\#} \times 100\%$, where $f^{\#} = \min\{f^{\text{ICA}}, f^{\text{GA}}\}$. Similarly, the RDP for GA is defined as $(f^{\text{GA}} - f^{\#})/f^{\#} \times 100\%$. For ICA, with the increasing number of developers, the run time also increases. Conversely, the run time of GA is slightly influenced by m. As expected, ICA always provides the minimal objective cost, *i.e.*, $f^{\#}$. For a 200-job project in the real world, it is worthwhile to wait for 200 s and obtain its near-optimal schedule.

In Table 9, ICA and GA are compared for larger problem instances, *i.e.*, 400. In general, the run time of ICA is slightly affected by the number of developers and unaffected by the distribution of developers. On the other hand, the number and the distribution of developers do not increase the run time of GA. However, the solution quality of GA deteriorates as the distribution of developers varies, *e.g.*, a large RDP of 20.576%.

Table 10 shows the influence of $n = 600$ on the performances of ICA and GA. It is interesting that both metaheuristic algorithms excel in processing equally-sized jobs, *i.e.*, $D^J = 0$. However, the problem becomes difficult for both algorithms if the jobs have different processing times. Although ICA takes 10 min on average, it generates better solutions in most situations. Moreover, in the real world, scheduling 600 jobs within 10 min is allowable.

TABLE 9. The performance of ICA for $n = 400$ and $D^J = 3$.

		ICA				GA			
		Run Time		RDP		Run Time		RDP	
m	D^D	Mean	Max	Mean	Max	Mean	Max	Mean	Max
2	0	320.264	400.735	0.000	0.000	3.343	7.031	1.174	2.394
	1	360.954	400.968	0.000	0.000	2.980	5.516	1.589	3.732
	2	379.643	401.015	0.000	0.000	2.992	6.625	1.318	2.529
	3	358.217	400.891	0.000	0.000	2.965	5.750	1.789	5.955
3	0	391.496	401.578	0.000	0.000	2.973	6.875	4.612	8.494
	1	394.137	401.437	0.000	0.000	3.217	6.047	5.162	9.960
	2	396.411	401.781	0.000	0.000	3.268	5.656	7.000	11.317
	3	393.300	401.828	0.000	0.000	2.844	4.640	9.656	19.388
5	0	395.284	401.797	0.000	0.000	3.199	6.015	6.903	11.393
	1	400.716	402.109	0.000	0.000	3.113	5.843	8.319	13.983
	2	400.676	401.640	0.000	0.000	3.398	7.844	12.559	23.537
	3	400.398	402.578	0.000	0.000	3.164	5.703	20.576	47.307

TABLE 10. The performance of ICA for $n = 600$, $m = 3$, and $D^D = 2$.

	ICA				GA			
	Run Time		RDP		Run Time		RDP	
D^J	Mean	Max	Mean	Max	Mean	Max	Mean	Max
0	35.397	51.515	0.000	0.000	3.396	3.515	0.000	0.000
1	351.333	600.422	0.140	2.802	6.943	18.610	1.042	2.941
2	590.282	602.328	0.000	0.000	4.076	8.078	5.962	26.819
3	601.092	602.328	0.000	0.000	4.736	10.469	7.358	16.987

In Figure 5, a sensitivity test shows the benefits of employing versatile developers. At the beginning, we make 5 mediocre developers process 100 jobs of various types. The estimated objective cost is 707.55. Then we replace the first developer with a versatile one at random. The objective cost is quickly reduced to 485.49. After repetition, the objective cost is only 182.85 if all the mediocre developers are replaced by versatile developers. Suppose that the salary of a versatile developer is 300% of that of a mediocre one. However, the throughput of a versatile developer is 386.96% higher than that of a mediocre one (= (1/182.85)/(1/707.55) = 386.96%). Clearly, it is worth paying a trilingual dubbing specialist triple the salary, instead of hiring three mediocre voice actors.

7. Discussion and conclusion

In this study, we present an interesting scheduling problem in the game industry. Three contributions are made. First, research findings show that versatile developers are not equivalent to efficient machines. For example, some machines which efficiently heat, extrude, and pull aluminum billets cannot spray paint the corresponding final products. That is, machines are usually dedicated for only one particular purpose. Developers, in contrast, might excel not only in figure modeling but also in programming. Therefore, new scheduling algorithms are called for. Second, an exact algorithm (B&B) is proposed to serve as a benchmark of solution quality, and a metaheuristic algorithm (ICA) is developed for obtaining approximate schedules in the real world. Third, a sensitivity test is conducted to differentiate between a versatile developer and a mediocre one.

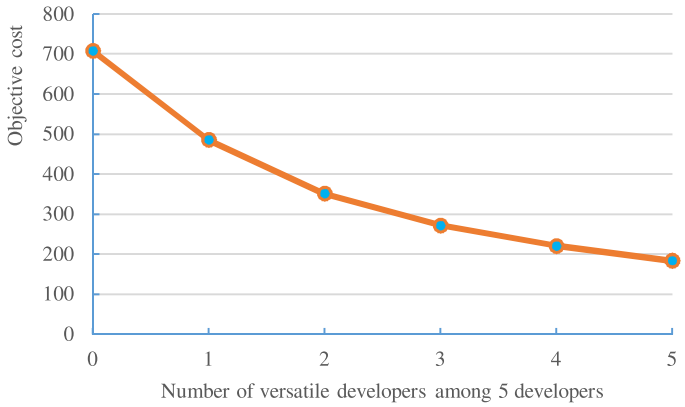


FIGURE 5. A sensitivity test for developers ($n = 100$, $m = 5$, $D^J = 3$).

Compared with past research, this study has the following features. First, traditional branch-and-bound algorithms for scheduling *unifunctional* machines cannot be directly applied to the presented problem. That is, the consideration of versatile developers makes this study more practical and realistic. Second, the proposed branch-and-bound algorithm is relatively efficient. For traditional machine scheduling problems, *e.g.*, [63], $n = 25$ is the maximal size for their branch-and-bound algorithms. Note that their machines are *identical* and all of them process jobs at a fixed pace; *i.e.*, they are easier problems. However, in this study, we must consider three kinds of jobs and m heterogonous developers. Consequently, $n = 12$ is an acceptable problem size for our B&B algorithm. Third, the solution quality of ICA is ensured, for B&B can generate the optimal solutions which we can use them as fair benchmarks to address the solution quality gap between ICA and B&B.

Although these proposed algorithms are relatively efficient, they still have some shortcomings. We may overcome these shortcomings by considering the following future directions.

- Some non-preemptive lower bounds are helpful for improving the efficiency of a branch-and-bound algorithm, for preemption may lead to underestimations of actual objective costs.
- Some mathematical analyses, *e.g.*, [31], can help a metaheuristic algorithm to accelerate its execution speed. That is, we can skip some invalid solutions and reduce the execution time.
- Hybridization may be beneficial for improving the efficiency of a metaheuristic algorithm. The related findings regarding lower bound may be valuable information for developing some operations, such as mutation.

Acknowledgements. This study was partially supported by the Ministry of Science and Technology of Taiwan, R.O.C. under Project MOST-110-2410-H-241-001. The authors thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this study.

REFERENCES

[1] L.R. Abreu, R.F. Tavares-Neto and M.S. Nagano, A new efficient biased random key genetic algorithm for open shop scheduling with routing by capacitated single vehicle and makespan minimization. *Eng. Appl. Artif. Intell.* **104** (2021) 104373.

[2] B. Alidaee and A. Ahmadian, Two parallel machine sequencing problems involving controllable job processing times. *Eur. J. Oper. Res.* **70** (1993) 335–341.

[3] M. Androvich, GTA IV: most expensive game ever developed? *Games Ind. Int.* **30** (2008).

[4] J.P. Arnaout, R. Musa and G. Rabadi, A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines-part II: enhancements and experimentations. *J. Intell. Manuf.* **25** (2014) 43–53.

[5] M.A. Bajestani and R.T. Moghaddam, A new branch-and-bound algorithm for the unrelated parallel machine scheduling problem with sequence-dependent setup times, in Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing. Vol. 42. Moscow, Russia (2009) 792–797.

[6] C. Blum and A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Comput. Surv.* **35** (2003) 268–308.
[7] B. Brittain, IN BRIEF: Tencent's Supercell hit with $92 million mobile-game patent verdict. Available: https://www.reuters.com/business/legal/brief-tencents-supercell-hit-with-92-million-mobile-game-patent-verdict-2021-05-10/ (2021).
[8] J.T. Chang, X.G. Kong and L. Yin, A novel approach for product makespan prediction in production life cycle. *Int. J. Adv. Manuf. Technol.* **80** (2015) 1433–1448.
[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithms, 3rd edition. MIT Press (2009).
[10] A. Costa, F.A. Cappadonna and S. Fichera, Minimizing the total completion time on a parallel machine system with tool changes. *Comput. Ind. Eng.* **91** (2016) 290–301.
[11] G. Divsalar, A. Divsalar, A. Jabbarzadeh and H. Sahebi, An optimization approach for green tourist trip design. *Soft Comput.* **26** (2022) 4303–4332.
[12] H.H. Doulabi and S. Khalilpourazari, Stochastic weekly operating room planning with an exponential number of scenarios. *Ann. Oper. Res.* (2022). DOI: 10.1007/s10479-022-04686-4.
[13] A. Ebrahimi, H.W. Jeon, S. Lee and C. Wang, Minimizing total energy cost and tardiness penalty for a scheduling-layout problem in a flexible job shop system: a comparison of four metaheuristic algorithms. *Comput. Ind. Eng.* **141** (2020) 106295.
[14] B. Fritz, Video Game Borrows Page from Hollywood Playbook. Los Angeles Times (2009).
[15] B. Fritz and A. Pham, Star Wars: The Old Republic – The Story Behind a Galactic Gamble. Los Angeles Times (2012).
[16] M. Ganji, H. Kazemipoor, S.M.H. Molana and S.M. Sajadi, A green multi-objective integrated scheduling of production and distribution with heterogeneous fleet vehicle routing and time windows. *J. Cleaner Prod.* **259** (2020) 120824.
[17] J.S. Gao, X.M. Zhu and R.T. Zhang, A branch-and-price approach to the multitasking scheduling with batch control on parallel machines. *Int. Trans. Oper. Res.* **2022** (2022) 1–22.
[18] M. Ghirardi and C.N. Potts, Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. *Eur. J. Oper. Res.* **165** (2005) 457–467.
[19] 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, New Jersey, USA (1985) 154–159.
[20] W.Y. Jia, Z.B. Jiang and Y. Li, Scheduling to minimize the makespan in large-piece one-of-a-kind production with machine availability constraints. *Expert Syst. App.* **42** (2015) 9174–9182.
[21] R. Jovanovic and S. Voss, Fixed set search application for minimizing the makespan on unrelated parallel machines with sequence-dependent setup times. *Appl. Soft Comput.* **110** (2021) 107521.
[22] I. Kacem and C.B. Chu, Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *Int. J. Prod. Econ.* **112** (2008) 138–150.
[23] V. Kayvanfar, G.M. Komaki, A. Aalaei and M. Zandieh, Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. *Comput. Oper. Res.* **41** (2014) 31–43.
[24] S. Khalilpourazari, Using reinforcement learning to forecast the spread of COVID-19 in France, in 2021 IEEE International Conference on Autonomous Systems (ICAS). Montreal, Canada (2021).
[25] S. Khalilpourazari and H.H. Doulabi, Robust modelling and prediction of the COVID-19 pandemic in Canada. *Int. J. Prod. Res.* **2021** (2021) 1–17.
[26] S. Khalilpourazari, H.H. Doulabi, A.O. Ciftcioglu and G.W. Weber, Gradient-based grey wolf optimizer with Gaussian walk: application in modelling and prediction of the COVID-19 pandemic. *Expert Syst. App.* **177** (2021) 114920.
[27] A. Khoudi and A. Berrichi, Minimize total tardiness and machine unavailability on single machine scheduling problem: bi-objective branch and bound algorithm. *Oper. Res.* **20** (2020) 1763–1789.
[28] E. Kozlowski, D. Mazurkiewicz, T. Zabinski, S. Prucnal and J. Sep, Machining sensor data management for operation-level predictive model. *Expert Syst. App.* **159** (2020) 1–10.
[29] M. Lanzetta, A. Rossi and A. Puppato, Modelling activity times by hybrid synthetic method. *Prod. Planning Control* **27** (2016) 909–924.
[30] 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.
[31] W.C. Lee and J.Y. Wang, A three-agent scheduling problem for minimizing the flow time on two machines. *RAIRO: Oper. Res.* **54** (2020) 307–323.
[32] W.C. Lee, J.Y. Wang and L.Y. Lee, A hybrid genetic algorithm for an identical parallel-machine problem with maintenance activity. *J. Oper. Res. Soc.* **66** (2015) 1906–1918.
[33] 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.
[34] T.K. Liang, B. Zeng, J.Q. Liu, L.F. Ye and C.F. Zou, An unsupervised user behavior prediction algorithm based on machine learning and neural network for smart home. *IEEE Access* **6** (2018) 49237–49247.
[35] Y.K. Lin and F.Y. Hsieh, Unrelated parallel machine scheduling with setup times and ready times. *Int. J. Prod. Res.* **52** (2014) 1200–1214.
[36] F.P.-C. Lin and F.K.H. Phoa, Runtime estimation and scheduling on parallel processing supercomputers via instance-based learning and swarm intelligence. *Int. J. Mach. Learn. Comput.* **9** (2019) 592–598.
[37] S.W. Lin and K.C. Ying, ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times. *Comput. Oper. Res.* **51** (2014) 172–181.

[38] H. Mokhtari, A nature inspired intelligent water drops evolutionary algorithm for parallel processor scheduling with rejection. *Appl. Soft Comput.* **26** (2015) 166–179.

[39] E. Molaee, R. Sadeghian and P. Fattahi, Minimizing maximum tardiness on a single machine with family setup times and machine disruption. *Comput. Oper. Res.* **129** (2021) 105231.

[40] M.E. Moore and J. Novak, Game Development Essentials: Game Industry Career Guide. Cengage Learning (2010).

[41] M. Moser, N. Musliu, A. Schaerf and F. Winter, Exact and metaheuristic approaches for unrelated parallel machine scheduling. *J. Scheduling* **25** (2021) 507–534.

[42] H.M. Motair, Exact and hybrid metaheuristic algorithms to solve bi-objective permutation flow shop scheduling problem, in Iraqi Academics Syndicate International Conference for Pure and Applied Sciences. Vol. 1818. Babylon, Iraq (2022) 1–10.

[43] S. Nayeri, Z. Sazvar and J. Heydari, A fuzzy robust planning model in the disaster management response phase under precedence constraints. *Operational Research* **22** (2022) 3571–3605.

[44] R. Nessah and I. Kacem, Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates. *Comput. Oper. Res.* **39** (2012) 471–478.

[45] R. Nessah, F. Yalaoui and C.B. Chu, A branch-and-bound algorithm to minimize total weighted completion time on identical parallel machines with job release dates. *Comput. Oper. Res.* **35** (2008) 1176–1190.

[46] O. Ozturk, M.A. Begen and G.S. Zaric, A branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize makespan. *Int. J. Prod. Res.* **55** (2017) 1815–1831.

[47] J. Pacheco, F. Angel-Bello and A. Alvarez, A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *J. Scheduling* **16** (2013) 661–673.

[48] Z. Pei, M.Z. Wan and Z.T. Wang, A new approximation algorithm for unrelated parallel machine scheduling with release dates. *Ann. Oper. Res.* **285** (2020) 397–425.

[49] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems. Springer, New York (2010).

[50] F.J. Rodriguez, M. Lozano, C. Blum and C. Garcia-Martinez, An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Comput. Oper. Res.* **40** (2013) 1829–1841.

[51] R. Rudek, The single processor total weighted completion time scheduling problem with the sum-of-processing-time based learning model. *Inf. Sci.* **199** (2012) 216–229.

[52] R. Rudek, A fast neighborhood search scheme for identical parallel machine scheduling problems under general learning curves. *Appl. Soft Comput.* **113** (2021) 108023.

[53] J.E. Schaller, Minimizing total tardiness for scheduling identical parallel machines with family setups. *Comput. Ind. Eng.* **72** (2014) 274–281.

[54] D. Senapati, A. Sarkar and C. Karfa, Performance-effective DAG scheduling for heterogeneous distributed systems, in ICDCN 2022: 23rd International Conference on Distributed Computing and Networking. Delhi, India (2022) 234–235.

[55] J.M.P. Silva, E. Teixeira and A. Subramanian, Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. *J. Oper. Res. Soc.* **72** (2021) 444–457.

[56] H. Singh, S. Tyagi, P. Kumar, S.S. Gill and R. Buyya, Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: analysis, performance evaluation, and future directions. *Simul. Modell. Pract. Theory* **111** (2021) 102353.

[57] H. Soleimani, H. Ghaderi, P.W. Tsai, N. Zarbakhshnia and M. Maleki, Scheduling of unrelated parallel machines considering sequence-related setup time, start time-dependent deterioration, position-dependent learning and power consumption minimization. *J. Cleaner Prod.* **249** (2020) 119428.

[58] C.H. Su and J.Y. Wang, A branch-and-bound algorithm for minimizing the total tardiness of multiple developers. *Mathematics* **10** (2022) 10071200.

[59] A. Subramanian, M. Battarra and C.N. Potts, An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Int. J. Prod. Res.* **52** (2014) 2729–2742.

[60] C.S. Sung and Y.I. Choung, Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *Eur. J. Oper. Res.* **120** (2000) 559–574.

[61] S. Tanaka and M. Araki, A branch-and-bound algorithm with Lagrangian relaxation to minimize total tardiness on identical parallel machines. *Int. J. Prod. Econ.* **113** (2008) 446–458.

[62] M.D. Toksari, A branch and bound algorithm for minimizing makespan on a single machine with unequal release times under learning effect and deteriorating jobs. *Comput. Oper. Res.* **38** (2011) 1361–1365.

[63] J.Y. Wang, Algorithms for minimizing resource consumption over multiple machines with a common due window. *IEEE Access* **7** (2019) 172136–172151.

[64] J.Y. Wang, A branch-and-bound algorithm for minimizing the total tardiness of a three-agent scheduling problem considering the overlap effect and environment protection. *IEEE Access* **7** (2019) 5106–5123.

[65] J.Y. Wang, Minimizing the total weighted tardiness of overlapping jobs on parallel machines with a learning effect. *J. Oper. Res. Soc.* **71** (2020) 910–927.

[66] S.J. Wang and M. Liu, A branch and bound algorithm for single-machine production scheduling integrated with preventive maintenance planning. *Int. J. Prod. Res.* **51** (2013) 847–868.

[67] J.Y. Wang, M.W. Chen and K.F. Jea, Minimizing the total tardiness of a game project considering the overlap effect. *IEEE Access* **8** (2020) 216507–216518.

[68] X. Wang, T. Ren, D. Bai, C. Ezeh, H. Zhang and Z. Dong, Minimizing the sum of makespan on multi-agent single-machine scheduling with release dates. *Swarm Evol. Comput.* **69** (2022) 100996.

[69] S.Q. Yao, Z.B. Jiang and N. Li, A branch and bound algorithm for minimizing total completion time on a single batch machine with incompatible job families and dynamic arrivals. *Comput. Oper. Res.* **39** (2012) 939–951.

[70] Y.Q. Yin, W.H. Wu, W.H. Wu and C.C. Wu, A branch-and-bound algorithm for a single machine sequencing to minimize the total tardiness with arbitrary release dates and position-dependent learning effects. *Inf. Sci.* **256** (2014) 91–108.

[71] M. Zandieh and M. Roumani, A biogeography-based optimization algorithm for order acceptance and scheduling. *J. Ind. Prod. Eng.* **34** (2017) 312–321.

[72] L.K. Zhang, Q.W. Deng, R.H. Lin, G.L. Gong and W.W. Han, A combinatorial evolutionary algorithm for unrelated parallel machine scheduling problem with sequence and machine-dependent setup times, limited worker resources and learning effect. *Expert Syst. App.* **175** (2021) 114843.