Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 8405961, 13 pages
https://doi.org/10.1155/2019/8405961

Research Article

Dynamic Multiobjective Software Project Scheduling Optimization Method Based on Firework Algorithm

Jian Cheng, Jianjiao Ji, Yi-nan Guo, and Junhua Ji

China University of Mining and Technology, Xuzhou 221116, China

Correspondence should be addressed to Yi-nan Guo; guoyinan@cumt.edu.cn

Received 25 January 2019; Accepted 4 June 2019; Published 1 July 2019

Academic Editor: Andrés Sáez

Software project scheduling is essentially a kind of project scheduling problem with limited human resources. During the development process of a software product, reworking the completed projects, reassessing the workload, and changing the number of employees or their skills are the frequently occurring dynamic issues having direct influences on designing a scheduling scheme. Taking the development cost and duration, the robustness, and the stability of the scheduling schemes as the objective functions, software project scheduling is modeled as a dynamic four-objective optimization problem. The various programming habits among the employees form the specific constraints for the reworking tasks, and the skills of the employees vary due to the effects of learning and forgetting. To solve this problem, an improved multiobjective firework algorithm with a novel explosion operator and reservation strategy is incorporated with rescheduling methods to fully guide the evolution by using the historical evolutionary knowledge. The experimental results indicate that the proposed method has better scheduling performance, and the optimal scheduling schemes have better robustness and stability.

1. Introduction

Efficiently developing and maintaining software products determines the market competitiveness of a software company. To meet the requirements of software development, such as deadlines and budgets, effectively allocating dozens of employees to cooperatively fulfill a project timeline subject to various constraints is a key issue, which is called the software project scheduling problem (SPSP) [1]. An inappropriate project schedule may result in project failure and even large economic losses.

With the increasing of the software's complexity, more employees are allocated to complete the numerous tasks. Scheduling the complicated software project manually or by traditional numerical programming is inefficient and possibly falls into an unfeasible project schedule [2, 3]. Moreover, many unpredictable events often occur in real-world software development processes, such as employees leaving and joining, changing the task requirements, and the arrival of patching tasks for the developed software, which change the working environment of a project. The employees, as a special scheduling resource, normally possess more than one kind of skill and their proficiencies change with experience due to the inherent learning and forgetting ability of human beings. The optimal schedule that is suitable for the certain development condition is usually unable to achieve the desired performance and may even deteriorate the performance under the changed environment. Based on this, employing the dynamic software project scheduling method is a necessity.

Given that dynamic events that occur in the development process directly affect the schedules, accurately describing the events and modeling the SPSP are two key issues for solving this NP-hard problem. Three kinds of unpredictable and uncertain events, including the uncertain task effort, the new task arrival, and employee leave and return, are first discussed and modeled by Shen [4]. However, the SPSP model with the above dynamic factors is incomplete due to various risks from customers, the development environment, and task demand that may happen in the software project [5]. For example, changing customer demands or wrong estimations regarding the product complexity are apt to modify the functions or cause bugs in software, which requires what is called a reworking task. Without loss of generality, employees that were previously responsible for developing this software are

Table 1: Basic symbols and their notations.

Symbol	Definition
$R = \{e_1, e_2, \ldots e_i, \ldots e_I\}$	The set of employees
$V = \{a_1, \ldots a_j, \ldots a_{J+2}\}$	The set of tasks
$Sk = \{sk_1, \ldots sk_k, \ldots sk_K\}$	The set of skills required by the tasks
d_j	The duration of task a_j
st_j	The start time of task a_j
P_{aj}	The predecessor set of a_j
$S = \{s_1, \ldots s_i, \ldots s_I\}$	The weekly salary of the employees for the project
s_i	Employee e_i per unit time salary
t_i	The total working time of employee e_i
e_i^k	Employee e_i's skill level for skill sk_k
a_j^k	Task a_j's workload for skill sk_k
Sch	The workload of the whole project
T_c	The costs of the software project
T_d	The duration of the software project
T_r	The robustness of the software project
T_s	The stability of the software project

assigned to the task because of their personal programming habits and the high priority of tasks. Understanding the original codes written by the other developers and repairing the functions are time-consuming and unreasonable. In addition, developing the software project in fact requires the mental labor of employees [6]. Employee leave and return changes the search space of the software project scheduling problem, and their skill proficiencies are not fixed in time due to the inherent learning and forgetting ability of human beings. Paiva and Keumseok [7, 8] believed that the learning and forgetting effect is important and accompanying factor affecting the software development process. Active learning, such as participating in the training class for a certain skill, unconscious learning that implicitly accumulates more experience from the development process, and forgetting effects obviously change their skill proficiencies [9] and development efficiency [10]. However, few studies had been done on the learning and forgetting effect of employees. Proactive scheduling and reactive scheduling are essentially the rescheduling method to solve the SPSP with dynamic events [11–14]. Shen [4, 15] presents a dynamic scheduling method based on Q-learning. Though dynamic employees' skills considering the learning and forgetting effect had been presented by the authors [16], the other dynamic factors, the robustness and the stability of the schedules were not taken into account in software project scheduling model. Based on this, taking the duration and cost of the project, the robustness, and the stability of the schedules as the objective functions, dynamic multiobjective software project scheduling model is constructed. In particular, reworking task arrivals, dynamic skill proficiencies, and employee leave and return are taken into consideration. To solve this problem, an improved multiobjective firework algorithm with a novel heuristic evolutionary operators is presented as the rescheduling method to fully utilize the historical evolutionary knowledge.

The rest of this paper is organized as follows. The model of the dynamic software project scheduling problem is established in Section 2. Section 3 presents the improved multiobjective firework algorithm to solve the dynamic SPSP. In Section 4, the experimental results for the instances are compared and analyzed. At last, the whole paper is concluded and our planned future work is given.

2. Dynamic Multiobjective Software Project Scheduling Model

The essence of software project scheduling is to fulfill the tasks with complex precedence relationship by coordinating the skills of numerous employees. Each task requires an amount of human resources, and its resources demand cannot exceed the total amount of available resources. Three time-varying events, including arriving reworking tasks, leaving and returning of employees, and changing skill proficiency, are taken into account. Newly arrived reworking tasks received prior to the development projects shall be completed by employees before new projects, and may possibly result in leaving of employees before the projects are completed. This decelerates the development process because all of the subsequent tasks that cannot be done in advance and reassigning other employees may be inefficient. Moreover, we also concern about an uncertain factor, inaccurately estimating work load. The workload of a task is usually estimated using the COCOMO model [17]. Unexpected events, such as changing customer requirements, may change the task workload over time.

The basic symbols and their notations for the dynamic SPSP model are shown in Table 1.

Without loss of generality, an entire software development project consists of $J + 2$ tasks. The first and the last one at the beginning and end of a project, which are expressed as a_1 and a_{J+2}, are dummy tasks. All tasks must be conducted

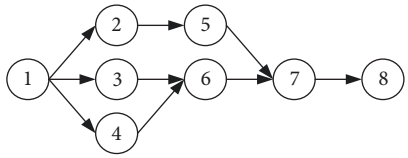


Figure 1: AON network.

in terms of the precedence relationship represented by the activity-on-node (AON) network, as shown in Figure 1. Task a_j can start up only after all tasks in Pa_j have been completed.

Fulfilling a task normally depends on one or more skills, except for the dummy tasks. Different tasks need various skills, and corresponding workloads also differ from each other. The task during its execution cannot be interrupted, and employees taking over a task cannot drop out. Hence, the duration of task a_j, which is denoted by d_j, is maximum makespan of all employees taking part in this task.

$$d_j = \left\{ \max \left\lceil \frac{a_j^k}{e_i^k} \right\rceil, k \in [1, K] \right\} \tag{1}$$

Each employee has a number of skills and their skill proficiencies may be different from other developers. At any moment, an employee can use a skill to complete only one task. Once an employee is assigned to a task, he (she) can be involved in another work task only after fulfilling the current one.

2.1. Modeling the Learning and Forgetting Effects. The skill proficiencies of employees may change with active or unconscious learning and forgetting. Previous studies on the software project scheduling problem [18, 19] normally assumed that the skill proficiency is fixed and less report on the learning and forgetting effects. Based on this, a model describing the time-varying skill proficiencies of employees with the learning and forgetting effect is built.

Wright's learning curve (WLC) proposed by Wright [20] first described the effect of the product quantity on employees' skill proficiencies in aircraft production. Given that learning is often accompanied by forgetting, as the inherent characteristics of human being, the relationship between employees' downtime and forgetting is subsequently described by variable regression to variable forgetting (VRVF) model [21]. Rationally utilizing the learning and forgetting effect can help project managers to fully know their employees' statuses and construct more efficient and reasonable schedules [22–24]. Being different from the traditional product scheduling problems, lines of code are normally employed to describe the workload changing with the learning/forgetting effect in the SPSP instead of the makespan of the unit product. Besides, once no task is fulfilled by the employees, corresponding skill level decreases with the forgetting effect and does not remain unchanged in the traditional learning model.

Suppose that e_i is ith employee and sk_k is kth skill. The skill proficiency of sk_k for e_i is expressed by e_i^{k}. Based on the WLC and VRVF models, the learning and forgetting model for the SPSP is presented as follows [16]:

$$e_i^k = \begin{cases} e_i^k|_{\underline{tw_{ij}^k}} \cdot x^{\alpha_i^k} & tw_{ij}^k \in Tw_i^k \\ e_i^k|_{\underline{ts_{ij}^k}} \cdot y^{-\beta_i^k} & ts_{ij}^k \in Ts_i^k \end{cases} \tag{2}$$

In the above formula, tw_{ij}^k represents the time consumed by e_i to complete task a_j with her (his) skill sk_k. If employee e_i does not spend any time on fulfilling this task with sk_k, the time is expressed by ts_{ij}^{k}. Tw_i^k and Ts_i^k, respectively, denote the total time that e_i uses the skill sk_k or not, and they satisfy $Tw_i^k \cap Ts_i^k = \Phi$ and $Tw_i^k \cup Ts_i^k = T_d$. $e_i^k|_{tw_{ij}^k}$ and $e_i^k|_{ts_{ij}^k}$ are, respectively, the skill proficiency of sk_k for e_i at times tw_{ij}^k and ts_{ij}^{k}. α and β, which are real numbers between $[0, 1]$, are the coefficients of learning and forgetting which are preset by the decision-makers. The starting points of learning or forgetting are not fixed. x and y are the workloads with or without tasks for the employees.

$$x = e_i^k\big|_{tw_{ij}^k} \cdot tw_{ij}^k (1 - \varphi) \tag{3}$$

$$y = e_i^k\big|_{ts_{ij}^k} \cdot ts_{ij}^k (1 - \varphi) \tag{4}$$

2.2. Modeling Dynamic Events. We focus on three characteristics of dynamic events, including their types, the times when events arrived, and their additional information, which are represented by el_1, el_2, and el_3, respectively.

$$ev = \{el_1, el_2, el_3\}$$

$$el_1 = \begin{cases} 1 & \text{reworking task} \\ 2 & \text{empolyee leave and return} \\ 3 & \text{reevaluation of task} \end{cases} \tag{5}$$

$$el_2 = t \tag{6}$$

$$el_3 = \begin{cases} \text{workload, required employees and priority of reworking task} & el_1 = 1 \\ \text{the number of the absent employees and the time he (she) returns} & el_1 = 2 \\ \text{reevaluation workload of the task} & el_1 = 3 \end{cases} \tag{7}$$

The probability of dynamic events occurring obeys a Poisson distribution $\lambda = T_d^{init}/2$. All dynamic events that occurred in a project consist of a dynamic event set expressed by Ev.

$$Ev = \{ev_1, ev_2, ev_3\} \tag{8}$$

2.3. Modeling Dynamic Software Project Scheduling Problem. Taking four objectives, including the duration and cost of the project, the robustness, and the stability of the schedules, into account, the dynamic software project scheduling problem is modeled as follows:

$$\min \{T_{\mathrm{d}}, T_{\mathrm{c}}, T_{\mathrm{r}}, T_{\mathrm{s}}\} \tag{9}$$

In the above formula, T_{d} represents the duration of the project. Given that the last task a_{j+2} is a dummy variable, the project is completed at its starting time, which is expressed by st_{J+2}. For each task, it starts after its preceding tasks have been accomplished. The start time of a_j is defined as follows:

$$T_{\mathrm{d}} = st_{J+2} \tag{10}$$

$$st_j = \max\left(st_m + d_m\right), \quad a_m \in P_{aj} \tag{11}$$

The cost of the project, which is denoted by T_{c}, depends on the salaries of employees and the time it takes to fulfill a task.

$$T_{\mathrm{c}} = \sum s_i t_i \tag{12}$$

T_{r} represents the robustness of the schedules for the SPSP with dynamic events. Once an unpredictable event arrives, the schedule with less change under the new environment has better robustness. Suppose that $T_{\mathrm{d}}^{\mathrm{init}}$ and $T_{\mathrm{c}}^{\mathrm{init}}$ are the duration and cost of a project before new dynamic events arrive. The duration and cost while dynamic events occur are denoted as T_{d}^q and T_{c}^q.

$$T_{\mathrm{r}} = \frac{1}{N_{\mathrm{d}}} \sum_{q=1}^{N_{\mathrm{d}}} \left| \frac{T_{\mathrm{d}}^q - T_{\mathrm{d}}^{init}}{T_{\mathrm{d}}^{init}} \right| + \lambda \frac{1}{N_d} \sum_{q=1}^{N_{\mathrm{d}}} \left| \frac{T_{\mathrm{c}}^q - T_{\mathrm{c}}^{init}}{T_{\mathrm{c}}^{init}} \right| \tag{13}$$

In the above formula, λ is a weight coefficient and is preset to 1. The set of all dynamic events that occur in a project are defined as $Ev_q \in D, q = 1, 2, \ldots, N_{\mathrm{d}}$ and the number of dynamic events is N_{d}. In this paper, $N_{\mathrm{d}} = 10$.

The stability of schedules reflects the differences of the employees' distributions under the dynamic environments, which is expressed by T_{s}.

$$T_{\mathrm{s}} = \frac{1}{N_{\mathrm{d}}} \sum_{q=1}^{N_{\mathrm{d}}} \sum_{j=1}^{J} \sum_{k=1}^{K} w_{jk} \Delta Sch_{jk}^q \tag{14}$$

$$\Delta Sch_{jk}^q = \begin{cases} 1 & \text{if } Sch^q(j,k) \neq Sch^{init}(j,k) \\ 0 & \text{if } Sch^q(j,k) = Sch^{init}(j,k) \end{cases} \tag{15}$$

In the above formula, $w_{jk} \in (0, 1]$ is the weight. Without loss of generality, an employee only can use a skill to fulfill the appropriate task, and the demand for a specific skill for a task only can be completed by an employee. Based on this, the constraints of the dynamic SPSP model are listed as follows:

$$\overline{y}_i = \frac{1}{T} \sum_{t=1}^{T_{\mathrm{d}}} y_{it} \tag{16}$$

$$\sum_{i=1}^{I} x_{ijk} \leq 1 \tag{17}$$

$$\sum_{j=1}^{J+2} \left(x_{ijk} y_{it}\right) \leq 1 \tag{18}$$

$$x_{ijk} = \begin{cases} 1 & e_i \text{ employs } sk_k \text{ in } a_j \\ 0 & \text{other} \end{cases} \tag{19}$$

$$y_i = \begin{cases} 1 & e_i \text{ is working at } t \\ 0 & \text{other} \end{cases} \tag{20}$$

3. Dynamic Multiobjective Software Project Scheduling Optimization Method Based on Firework Algorithm

The fireworks algorithm, which was first proposed by Tan [25], is derived from the explosion process of fireworks. Each firework or spark can be regarded as a feasible solution. The new individuals, which are called sparks, are generated by the exploding fireworks in the potential region. Only the sparks with the larger fitness values were selected as the explosion locations in the next generation. Two kinds of evolutionary operators, including the explosion operator and the mutation operator, are employed to generate the offspring sparks. The balance between the diversity and distribution is kept by controlling the explosion radius of the fireworks and the number of generated sparks, which are set in advance before

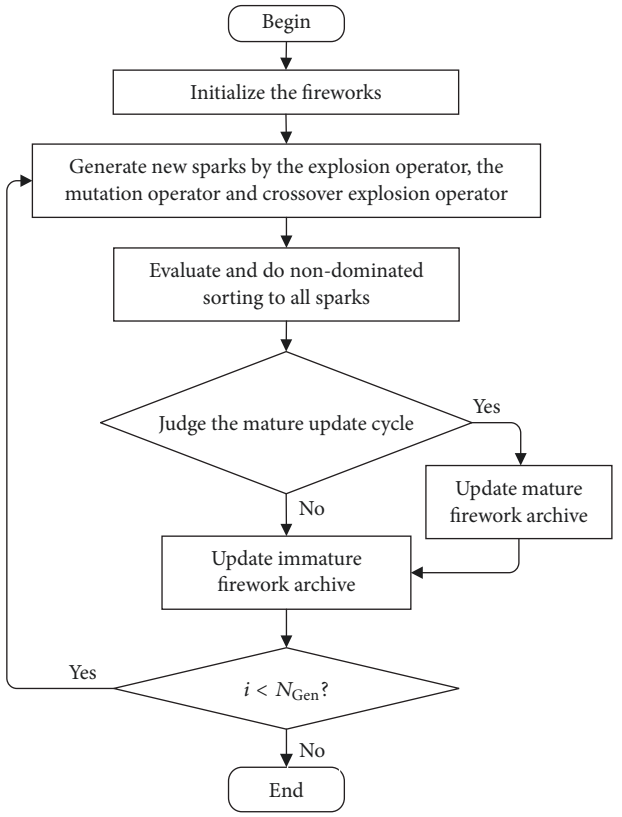


Figure 2: Flow chart of IFA.

the evolution. The above operators are employed to solve continuous optimization problems.

Given that the dynamic software project scheduling problem is a combination of optimization problems, different encodings of individuals cause various evolutionary operators. Based on this, an improved multiobjective firework algorithm (IFA) is proposed and the rough framework is depicted as follows (Figure 2). Here, i is the generation and N_{Gen} is the terminal iteration.

3.1. Encoding Method. The traditional binary encoding cannot describe the complicated relationship between the tasks and the employees, resulting in the infeasible solutions. The real-number encoding, consequently, is employed to construct a schedule. Without loss of generality, determining the executed order of tasks and assigning the employees are two necessary parts of a scheduling scheme, denoted as C_1 and C_2, respectively.

Randomly allocating the tasks is infeasible due to their precedential relationship. Thus, a rank-priority is employed to encode the tasks. Each task is assigned a rank in terms of their precedential relationship, and the ranks of the first and last dummy tasks are 1 and $J + 2$, respectively. A task having a lower rank must be executed and assigned the employees prior to others with higher ranks. Taking the AON network shown in Figure 1 as an example, each task, except for the dummy tasks (a_1 and a_8), needs four kinds of skills and eight employees possess these skills. The rank for all

Figure 3: Task allocation.

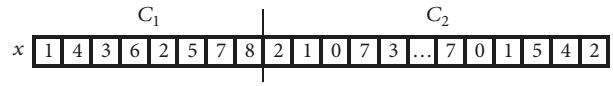


Figure 4: The vector of the employee assignment C_2.

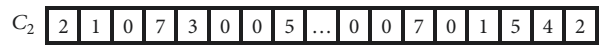


Figure 5: A software project scheduling scheme.

tasks is $\{1, 5, 3, 2, 6, 4, 7, 8\}$, and $C_1 = \{1, 4, 3, 6, 2, 5, 7, 8\}$ is obtained by arranging the tasks in ascending order, as shown in Figure 3.

An employee is assigned to fulfill a task in terms of its requirement for the skills. We denote J as the number of nondummy tasks in a project. The assignment scheme of an employee is defined as C_2 shown in Figure 4, except for the first and last dummy tasks, with the purpose of facilitating the evolutionary operations. By combining the task allocation scheme C_1 with the employee assignment scheme C_2, a complete software project scheduling scheme expressed by x is formed, as shown in Figure 5 [26].

3.2. Explosion Operator. Multiple sparks are produced by the explosion operator around a firework. Appropriately setting the explosion radius and the number of produced sparks is of importance. A nondominated firework produces more sparks within a small explosion radius with the purpose of enhancing the local search and exploitation abilities. A large explosion radius is employed to generate fewer sparks around the suboptimal firework to explore the search space with good diversity.

Given that a firework or spark is encoded by a vector composed of two parts, two explosion radii expressed by A_{n_c1} and A_{n_c2} are adaptively adjusted in terms of the ranks of individuals. The number of produced sparks for the nth individual also varies with their distribution.

$$A_{n\_c1} = \dot{A}_{c1} \frac{x_{n\_rank}}{\sum_{n=1}^{N} x_{n\_rank}} \tag{21}$$

$$A_{n\_c2} = \dot{A}_{c2} \frac{x_{n\_rank}}{\sum_{n=1}^{N} x_{n\_rank}} \tag{22}$$

$$S_n = M \frac{rank_{\max} - x_{n\_rank}}{\sum_{n=1}^{N} \left(rank_{\max} - x_{n\_rank}\right)} \tag{23}$$

In the above formulas, $rank_{\max} = \max(x_{n\_rank}) + 1$, $n = 1, 2, \ldots N$. $\dot{A}_{c1}$, $\dot{A}_{c2}$, and M are constants. x_{n_rank} represents the rank of x_n obtained by the nondominated sorting method [27]. We call $x_i \prec x_j$ if $(x_{i\_rank} < x_{j\_rank})$ or $(x_{i\_rank} = x_{j\_rank}$ and $x_{i\_crowd} > x_{j\_crowd})$.

Input: The sorted sparks set $S(t)$, previous mature firework archive $A_M(t-1)$
Output: $A_M(t)$
(1) if $A_M(t-1) = \varphi$
(2) $A_M(t)$ = the former N_M sparks of $S(t)$
(3) else
(4) S_{tmp} = Fast non-dominated sorting($A_M(t-1)\text{US}(t)$)
(5) S_N = the former N sparks of S_{tmp}
(6) $A_M(t) = S_N \cap A_M(t-1)$
(7) $S_{noAm} = S_{\text{tmp}} - A_M(t)$
(8) end if

Algorithm 1: Updating the mature firework archive.

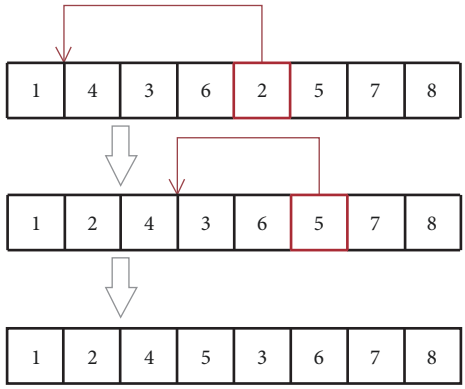


Figure 6: Explosion operator of C_1 by changing the priority of tasks.

Input: $S(t), S_{noAm}$
Output: $A_{IM}(t)$
(1) Sorting S_{noAm}
(2) $A_{Im\_a} \longleftarrow x_i(t) \in S_{noAm}$, $(i = 1, 2, \ldots N/2)$
(3) $S_{rest} = S_{noAm} - A_{Im\_a}$
(4) $A_{Im\_b} = \varphi$
(5) for $i = 1$ to $N/2$ do
(6) Sorting S_{rest}
(7) Random select 2 sparks from S_{rest}
(8) $A_{Im\_b} \longleftarrow x_1 \mid x_1 \prec x_2$
(9) $A_{Im\_b} \longleftarrow x_2 \mid x_1 \succ x_2$
(10) $A_{Im\_b} \longleftarrow x_1, x_2 \mid x_1 \parallel x_2$
(11) end for
(12) $A_{Im}(t) = A_{Im\_a} + A_{Im\_b}$

Algorithm 2: Updating the immature archive.

As shown in Figure 6, a new task allocation satisfying the precedential constraint is obtained by changing the priority of certain tasks or the sequence of tasks in parallel. For example, the priority of a_2 becomes lower after the explosion, and the corresponding task is completed first.

An employee is replaced by another one after randomly selecting a location in C_2 with a certain probability, which is expressed by r_{ijk}. Different from the traditional explosion operator, the probability depends on the model and the skill proficiencies of employees instead of a Gaussian distribution. An employee having the better skill proficiency is more likely to be selected.

$$r_{ijk} = \frac{e_i^k \times w_i^j}{\sum_{i=1}^{I} \left(e_i^k \times w_i^j\right)} \tag{24}$$

$$w_i^j = \begin{cases} 0 & e_i \text{ takes over } a_j \\ 1 & \text{else} \end{cases} \tag{25}$$

3.3. Reservation Strategy. The explosion operator is employed to explore the region around a firework, and it possibly falls into the local optimum, which causes an ineffective search and large computation cost. In the traditional reservation strategy of the firework algorithm, an individual having the large crowding distance is possibly a local optimum. However, calculating the crowding distance for the dynamic SPSP is meaningless because of its special encoding method. Therefore, a novel reservation strategy is proposed.

All sparks are divided into mature and immature fireworks and saved in different archives, which are denoted by A_M and A_{Im}, respectively. Individuals who are close enough to the local optimum are stored in A_M. The nondominated individuals out of A_M are saved as the sparks in A_{Im} who execute the explosion operation, the mutation operation, and the crossover operation in every *Ta* generation and the size of the mature archive is N_M. The detailed update processes of mature and immature archives are shown in Algorithms 1 and 2, respectively.

3.4. Heuristic Evolutionary Operators. Different mutation operators are applied to two parts of sparks. For C_1 of a randomly selected spark, x_n, changing the priority of any task results in a new C_1. An employee that is randomly selected from C_2 is replaced by another one having the same skill and satisfying the constraint for the workload of this skill. Moreover, the crossover operator is introduced to produce two sparks along the potential direction with the purpose of fully exchanging information between nondominated sparks in Algorithm 3.

Input: Immature firework archive A_{Im}
Output: new sparks
(1) Randomly select 2 fireworks x_1 and x_2 from A_{Im}
(2) Split x_1 and x_2 into x_{1a}, x_{1b}, x_{2a}, and x_{2b} respectively
(3) Form new sparks $x_{1\_new} = (x_{1a}, x_{2b})$ and $x_{2\_new} = (x_{2a}, x_{1b})$.

Algorithm 3: Crossover operator.

Table 2: Employee attributes.

Employee	Skill proficiency				Salary	α	β	φ
	sk_1	sk_2	sk_3	sk_4				
e_1	2	2	2	0	240	0.290	0.243	6.64‰
e_2	2	0	3	2	240	0.184	0.199	7.91%
e_3	3	0	0	1	240	0.236	0.228	8.24%
e_4	1	1	0	2	240	0.264	0.190	4.75%
e_5	0	3	1	0	240	0.324	0.220	3.66%
e_6	0	5	5	2	360	0.446	0.164	4.83%
e_7	0	5	0	4	360	0.358	0.177	1.54%
e_8	2	0	5	2	360	0.335	0.193	2.16%
e_9	0	3	5	0	360	0.325	0.151	0.35%

Table 3: Upper and lower limits of employees' skill proficiency.

Employee	Lower limit				Upper limit			
	sk_1	sk_2	sk_3	sk_4	sk_1	sk_2	sk_3	sk_4
e_1	1	2	1	0	4	4	4	0
e_2	1	0	1.5	1	4	0	4	4
e_3	2	0	0	1	4	0	0	4
e_4	1	1	0	1	4	4	0	4
e_5	0	2	1	0	0	4	4	0
e_6	0	3	2.5	1	0	6	6	6
e_7	0	3	0	2	0	6	0	6
e_8	1	0	2.5	1	6	0	6	6
e_9	0	2.5	3	0	0	6	6	0

Once a dynamic event arises, a heuristic strategy is designed for the dynamic software project scheduling problem to reschedule the employees.

For new reworking tasks that arrive prior to the executing one, employees participating in developing the original rework project are assigned to complete the urgent reworking task and replaced by other idle employees having the same skills to take over the suspended executing task. The mutation operator for C_2 is triggered to find the one that is most suitable for the executing software project.

The leave and return of employee are paired events, and only the tasks accomplished during this period shall be rescheduled. The idle employees having the highest skill proficiencies are arranged to complete the tasks according to the mutation operator for C_2. If no one is free, the task is delayed until any employee is available or the employee returns to perform the task.

4. Experimental Results and Analysis

The dynamic software project scheduling problem with three types of unpredictable events and unfixed skill proficiencies is built to further analyze the reasonableness of the proposed scheduling method. The 30 instances are divided into three groups with 10, 20, and 30 nondummy tasks that are obtained by using the generator of Alba and Chicano [28] and 3 other real instances are recorded as Real_b1, Real_2, and Real_3, respectively. The key parameters about employees are listed in Tables 2 and 3. Each kind of dynamic event occurs once at most in a project. All experiments are done by using MATLAB 7.0.

Three metrics are employed to analyze the performance of the schemes. The inverted generational distance (IGD) [29] calculates the distance between the objective values of the Pareto optimal solutions and the true Pareto front.

Table 4: Comparison of the Pareto optimal fronts obtained by two methods.

Instance	C(IFA-R,IFA-N) versus C(IFA-N,IFA-R)		Imp_i (IFA-R,IFA-N)			
	p-value	sign	T_c	T_d	T_s	T_r
T10_1	2.09E-4	+	0.31%	0.66%	3.56%	2.40%
T10_2	0.0420	+	2.22%	2.97%	2.64%	0.26%
T10_3	3.53E-6	+	0.64%	1.70%	5.49%	2.78%
T10_4	0.0589	=	*-1.21%*	*-2.42%*	6.34%	0.53%
T10_5	5.54E-6	+	1.04%	0.57%	3.79%	1.62%
T10_6	7.13E-4	+	2.99%	*-5.34%*	4.58%	4.61%
T10_7	0.17600	=	*-1.60%*	0.61%	4.43%	2.29%
T10_8	0.00753	+	*-0.60%*	0.96%	0.92%	3.46%
T10_9	0.00685	+	1.33%	2.21%	3.55%	0.24%
T10_10	1.14E-5	+	1.81%	2.26%	1.29%	1.41%
T20_1	2.43E-8	+	6.44%	2.52%	3.56%	7.98%
T20_2	4.29E-4	+	5.54%	7.90%	2.64%	6.34%
T20_3	0.00299	+	*-2.07%*	1.77%	5.49%	2.21%
T20_4	7.16E-4	+	5.70%	13.07%	6.34%	3.98%
T20_5	0.06210	=	*-2.60%*	*-0.12%*	3.79%	2.99%
T20_6	4.09E-5	+	2.52%	*-4.27%*	4.58%	4.83%
T20_7	0.00860	+	3.50%	5.77%	4.43%	1.14%
T20_8	2.53E-5	+	1.10%	3.56%	0.92%	3.31%
T20_9	3.58E-4	+	*-0.09%*	5.26%	3.55%	5.40%
T20_10	0.0647	=	*-1.49%*	*4.70%*	1.29%	1.19%
T30_1	0.0121	+	15.82%	17.93%	3.56%	9.84%
T30_2	0.0115	+	5.43%	3.38%	2.64%	5.37%
T30_3	8.21E-4	+	8.38%	14.93%	5.49%	13.31%
T30_4	0.25300	=	6.78%	*-4.63%*	6.34%	3.76%
T30_5	0.04040	+	3.53%	7.87%	3.79%	5.16%
T30_6	5.35E-4	+	0.02%	*-1.38%*	4.58%	6.85%
T30_7	9.51E-4	+	1.80%	3.84%	4.43%	12.70%
T30_8	5.57E-4	+	3.53%	*-5.21%*	0.92%	3.39%
T30_9	0.03400	+	*-0.41%*	*-0.06%*	3.55%	4.32%
T30_10	0.03900	+	9.57%	13.23%	1.29%	3.94%
Real_1	9.91E-5	+	3.52%	*-0.97%*	0.46%	5.12%
Real_2	3.61E-4	+	9.07%	11.3%	2.19%	8.34%
Real_3	0.00881	+	5.18%	6.62%	3.67%	12.7%

The hypervolume (HV) [30] represents the volume of the hypercube between the Pareto-optimal front and the reference point in objective space. The reference point is set to $[1, 1]$ after normalization. Spacing (SP) is employed to evaluate the distribution of the Pareto-optimal front. All of them measure the convergence and diversity of the Pareto optimal front. A lower IGD and larger HV mean that the optimal solutions better approximate the true Pareto front. Less SP means that individuals are evenly distributed along the Pareto-optimal front. In addition, C-metric and *Imp* are employed to compare the dominating relationship and the mean of each objective between the two methods [31].

4.1. Experiment I: The Influence of Robustness and Stability. To analyze the influence of the robustness and stability on the search direction, two methods with/without robustness and stability as the objectives, which are called the IFA-R and the IFA-N, are employed. Ten independent experiments are conducted for each instance. The Pareto-optimal fronts obtained by the two methods are compared in Table 4. The p-value calculated in the rank-sum test is assessed with respect to p=0.05. Here, "=" means that there is no obvious difference between the algorithm performances of the IFA-R and the IFA-N, while "+" indicates that the IFA-R has better performance than the IFA-N.

The p-value is less than 0.05 in 25 instances, which indicates that the IFA-R has significantly better performance than the IFA-N. The overall performance of the Pareto-optimal front skews the results in favor of taking the robustness and stability of the solutions into consideration. The italic part in the table labeled the degraded values of the IFA-R that are

Table 5: Comparison of algorithm performances among IFA, FA, NSGA-II, and MPSO.

Instance	IFA		FA		NSGA-II		MPSO		Imp_i(IFA, FA)		Imp_i(IFA, NSGA-II)		Imp_i(IFA, MPSO)	
	T_d	T_c	T_d	T_c	T_d	T_c	T_d	T_c	T_d	T_c	T_d	T_c	T_d	T_c
T10_1	24	19200	31	24360	26	21240	26	21341	28.33%	26.88%	8.33%	10.63%	8.33%	11.15%
T10_2	25	19392	31	24360	27	21396	27	23656	23.20%	25.62%	8.00%	10.33%	8.00%	21.99%
T10_3	24	20040	30	24144	26	21600	27	23775	23.33%	20.48%	8.33%	7.78%	12.50%	18.64%
T10_4	26	21240	29	23856	26	20640	29	24059	11.54%	12.32%	*0.00%*	*-2.82%*	11.54%	13.27%
T10_5	24	19200	30	24120	26	21000	28	22229	26.67%	25.63%	6.25%	9.38%	16.67%	15.78%
T10_6	24	19296	32	24312	27	21900	25	22140	31.67%	26.00%	10.42%	13.50%	4.17%	14.74%
T10_7	24	19200	30	23760	26	21240	28	23043	25.83%	23.75%	8.75%	10.63%	16.67%	20.02%
T10_8	24	19200	31	24072	25	20160	26	22960	29.17%	25.38%	2.92%	5.00%	8.33%	19.58%
T10_9	24	19200	29	24216	25	20160	27	21506	22.50%	26.13%	2.92%	5.00%	12.50%	12.01%
T10_10	24	19200	30	24504	27	22013	27	24653	24.17%	27.63%	11.83%	14.65%	12.50%	28.40%
T20_1	32	41184	58	62328	37	46620	40	47756	80.75%	51.34%	13.35%	13.20%	25.00%	15.96%
T20_2	36	41604	54	61704	43	47760	39	48521	50.56%	48.31%	19.50%	14.80%	8.33%	16.63%
T20_3	36	42816	55	60960	36	46718	42	48332	55.62%	42.38%	2.42%	9.11%	16.67%	12.88%
T20_4	35	43776	53	57648	40	48650	43	49010	51.00%	31.69%	14.07%	11.13%	22.86%	11.96%
T20_5	36	39888	58	60240	37	47940	42	51535	62.12%	51.02%	3.06%	20.19%	16.67%	29.20%
T20_6	34	41376	57	59232	35	48062	40	48173	67.46%	43.16%	3.49%	16.16%	17.65%	16.43%
T20_7	31	39456	53	59712	39	46080	41	49276	70.23%	51.34%	26.21%	16.79%	32.26%	24.89%
T20_8	32	43020	56	60696	37	50616	41	50984	76.58%	41.09%	17.22%	17.66%	28.13%	18.51%
T20_9	38	43056	55	61944	38	47376	41	49017	44.44%	43.87%	0.95%	10.03%	7.89%	13.84%
T20_10	33	39228	53	60696	38	45953	38	46276	59.04%	54.73%	15.72%	17.14%	15.15%	17.97%
T30_1	38	50292	61	73272	42	55920	42	57341	61.17%	45.69%	11.70%	11.19%	10.53%	14.02%
T30_2	41	50412	66	72168	46	57274	47	57747	58.94%	43.16%	12.17%	13.61%	14.63%	14.55%
T30_3	39	52284	58	68736	45	53760	46	55652	47.58%	31.47%	14.50%	2.82%	17.95%	6.44%
T30_4	43	53688	60	70344	44	57987	44	57389	37.64%	31.02%	1.57%	8.01%	2.33%	6.89%
T30_5	43	52872	60	69744	43	55114	46	57926	39.68%	31.91%	*-0.97%*	4.24%	6.98%	9.56%
T30_6	41	50340	61	71136	43	56030	45	56853	47.80%	41.31%	5.07%	11.30%	9.76%	12.94%
T30_7	37	48492	61	72792	40	55910	42	57623	66.30%	50.11%	8.91%	15.30%	13.51%	18.83%
T30_8	40	49476	62	72696	41	59326	40	58666	56.96%	46.93%	4.96%	19.91%	0.00%	18.57%
T30_9	40	50400	63	73320	45	56520	46	58847	56.82%	45.48%	11.66%	12.14%	15.00%	16.76%
T30_10	39	49824	64	69696	41	51840	43	54541	63.59%	39.88%	4.62%	4.05%	10.26%	9.47%
Real_1	24	26134	34	35801	25	29505	27	32684	30.18%	24.70%	6.40%	8.35%	12.50%	25.06%
Real_2	42	49809	66	72576	47	56925	48	58968	48.45%	54.32%	15.35%	17.74%	14.29%	18.39%
Real_3	51	59956	58	83956	56	67405	59	70081	43.70%	46.98%	14.10%	16.68%	15.69%	16.89%

worse than those of the IFA-N and it shows the four objectives coupled with each other. In summary, the robustness and stability of the nondominated solutions obtained by the IFA-R are significantly improved, while their costs and duration may be slightly worse.

4.2. Experiment II: Comparison of Algorithm Performances with Other Methods. The algorithmic performance of IFA is compared with those of the NSGA-II, the FA, and MPSO by running each instance 10 times, as shown in Tables 5 and 6. The nonpositive values in Table 5 have been in italic, and the optimal value of each metric in Table 6 is labeled in italic.

The means of the cost and duration are obviously optimized for 33 instances by IFA compared with the FA, MPSO, and NSGA-II. HV and IGD of IFA have the best performances, except for T10_4, meaning that IFA performs better with respect to the convergence and diversity. Compared with NSGA-II, MPSO, and FA, it indicates that the crossover operator and the novel reservation strategy significantly improve the performance of IFA. In addition, IFA has a better distribution due to the less SP for most of the instances. With the increasing number of the tasks in instances, the distribution of the Pareto-optimal front obtained by IFA becomes more uniform. Particularly, IFA shows the best performance in 3 real instances.

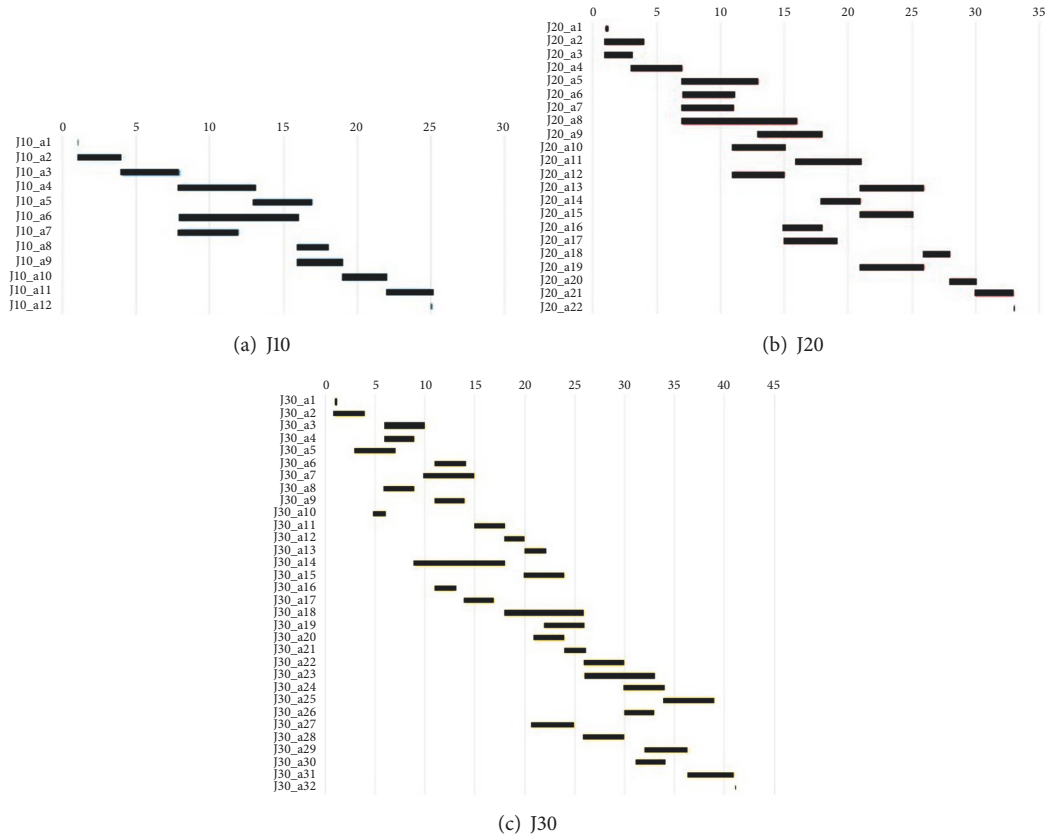
Table 6: Comparison of HV, IGD, and SP among IFA, FA, NSGA-II, and MPSO.

Instance	HV				IGD				SP			
	IFA	FA	NSGA-II	MPSO	IFA	FA	NSGA-II	MPSO	IFA	FA	NSGA-II	MPSO
T10_1	*0.0544*	0.0097	0.0248	0.0163	*0.0997*	0.6762	0.3254	0.4657	*0.0614*	0.1375	0.0928	0.1244
T10_2	*0.0455*	0.0055	0.0257	0.0162	*0.0898*	0.7535	0.4056	0.7053	*0.0614*	0.1337	0.1263	0.0979
T10_3	*0.0454*	0.0156	0.0220	0.0181	*0.1125*	0.5941	0.3668	0.3409	*0.0827*	0.1503	0.1263	0.1412
T10_4	0.0259	0.0149	*0.0289*	0.0191	0.2721	0.4681	*0.2616*	0.3819	0.1095	0.1831	*0.1006*	0.1133
T10_5	*0.0550*	0.0073	0.0336	0.0197	*0.2034*	0.6799	0.2820	0.3228	*0.0614*	0.1539	0.1236	0.0951
T10_6	*0.0534*	0.0066	0.0204	0.0165	*0.1846*	0.7512	0.4267	0.3180	*0.0295*	0.0826	0.1515	0.1539
T10_7	*0.0550*	0.0141	0.0241	0.0183	*0.1493*	0.5652	0.3254	0.6847	*0.0295*	0.1375	0.1686	0.1476
T10_8	*0.0536*	0.0077	0.0440	0.0179	0.1431	0.7299	*0.1286*	0.5399	*0.1089*	0.1520	0.1227	0.1150
T10_9	*0.0541*	0.0127	0.0432	0.0161	*0.1286*	0.5706	0.1444	0.5249	*0.0318*	0.1648	0.0694	0.1222
T10_10	*0.0540*	0.0097	0.0187	0.0174	*0.1272*	0.6431	0.4853	0.3225	*0.0318*	0.1443	0.1364	0.1187
T20_1	*0.1909*	0.0183	0.1137	0.0865	*0.0528*	0.7129	0.2282	0.6765	*0.0425*	0.0614	0.0481	0.0847
T20_2	*0.1586*	0.0129	0.0836	0.1118	*0.0910*	0.7534	0.3502	0.5610	*0.0526*	0.0615	0.0658	0.0681
T20_3	*0.1667*	0.0182	0.1148	0.0925	*0.0851*	0.7094	0.2298	0.4255	*0.0559*	0.0713	0.0673	0.0851
T20_4	*0.1511*	0.0403	0.0971	0.1011	*0.1333*	0.5797	0.3168	0.5066	*0.0271*	0.0879	0.0403	0.0906
T20_5	*0.1762*	0.0300	0.1120	0.0866	*0.0743*	0.6184	0.2658	0.4509	*0.0288*	0.0700	0.0567	0.0661
T20_6	*0.1681*	0.0240	0.1152	0.1041	*0.0605*	0.5964	0.3294	0.2880	0.0475	0.0683	*0.0425*	0.0609
T20_7	*0.2022*	0.0312	0.1008	0.0905	*0.0951*	0.6216	0.2481	0.3700	*0.0452*	0.0879	0.0637	0.0698
T20_8	*0.1734*	0.0346	0.0908	0.1062	*0.1016*	0.5713	0.3233	0.3117	*0.0382*	0.0753	0.0630	0.0709
T20_9	*0.1460*	0.0284	0.1170	0.1076	*0.1406*	0.5819	0.2439	0.3420	*0.0395*	0.0500	0.0429	0.0762
T20_10	*0.1877*	0.0372	0.1072	0.1099	*0.0197*	0.5930	0.2361	0.3700	*0.0398*	0.0640	0.0482	0.0804
T30_1	*0.1819*	0.0328	0.0972	0.0780	*0.0714*	0.6037	0.2295	0.3674	*0.0416*	0.0675	0.0600	0.0593
T30_2	*0.1526*	0.0359	0.0787	0.0634	*0.0709*	0.5817	0.3125	0.4266	*0.0447*	0.0704	0.0665	0.0681
T30_3	*0.1529*	0.0377	0.0915	0.0692	*0.0444*	0.5444	0.2391	0.6606	*0.0484*	0.0723	0.0672	0.0951
T30_4	*0.1457*	0.0389	0.1265	0.0965	*0.1605*	0.5485	0.1785	0.2577	*0.0430*	0.0613	0.0552	0.0565
T30_5	*0.1429*	0.0331	0.1016	0.0661	*0.1100*	0.5783	0.2224	0.2715	*0.0431*	0.0731	0.0510	0.1014
T30_6	*0.1558*	0.0326	0.1068	0.0930	*0.0662*	0.6398	0.2330	0.3345	0.0499	0.0635	*0.0416*	0.0915
T30_7	*0.1778*	0.0200	0.1309	0.0815	*0.0118*	0.6323	0.2151	0.5746	*0.0525*	0.0652	0.0629	0.0794
T30_8	*0.1690*	0.0318	0.0976	0.0999	*0.0467*	0.6140	0.2972	0.6159	*0.0456*	0.0871	0.0615	0.0839
T30_9	*0.1522*	0.0188	0.0814	0.0631	*0.0431*	0.6942	0.2828	0.5739	*0.0433*	0.0731	0.0495	0.0669
T30_10	*0.1634*	0.0360	0.1274	0.0777	*0.0336*	0.6064	0.1513	0.4755	*0.0523*	0.0761	0.0603	0.0779
Real_1	*0.1058*	0.0206	0.0706	0.0460	*0.1078*	0.6106	0.2487	0.4342	*0.04457*	0.1098	0.0666	0.0748
Real_2	*0.1583*	0.0308	0.0992	0.0404	*0.1204*	0.6694	0.2709	0.5628	*0.04062*	0.1027	0.0604	0.0734
Real_3	*0.1656*	0.0336	0.1107	0.0620	*0.0981*	0.6871	0.2690	0.6401	*0.04668*	0.0997	0.0687	0.1044

4.3. Experiment III: Influence of Dynamic Employees' Skill Proficiency. Three nondominated solutions for the instances with different numbers of the tasks are chosen to analyze the learning and forgetting effect on the skill proficiency by using the Gantt charts shown in Figure 7.

In the early stage of the project, the completion time of each task is relatively similar, and becomes longer over time, such as J10_a6, J20_a8, J30_a14, and J30_a19. In addition, the completion time of the task tends to be stable. Under the above situation, the curves of the skill proficiency changes over time are shown in Figure 8. The red, green, blue, and black curves represent *sk*1, *sk*2, *sk*3, and *sk*4, respectively.

The time consumption of J10_a6 is large because the workload of sk_1 for a_6 in instance J10 is done by e_2. The skill proficiency of sk_1 plotted by the red line in Figure 8 decreases at first due to the forgetting effect. Following that, e_2 is assigned to fulfill the task with this skill, causing the increasing of its skill proficiency because of the subconscious learning ability. The learning factor of e_2 is less than those of other employees, eventually leading to a longer time to fulfill the corresponding task. sk_4 of e_2 in J20_a8 and sk_2 of e_1 in J30_a14 have similar changing trends. Different from the former, the reason for the time consumption of J30_a19 is the competition of employees in tasks. The workloads of sk_2 and sk_4 for a_{19} in instance J30 are completed by e_2 and e_4, respectively. Actually, e_4 finished his (her) job in advance, but, at this time, e_2 is participating in another task and finally starts the workload of sk_4 for a_{19} later. In summary, the learning and forgetting effect of employees



(a) J10 (b) J20

(c) J30

Figure 7: Gantt chart of nondominated solutions.

have significant impacts on the development of the software project. Therefore, it is crucial to make reasonable assignments according to the dynamic characteristics of employees.

5. Conclusion

Various unpredictable events may arise in the development of a software project. In this paper, the dynamic software project scheduling problem with 3 types of frequently occurring dynamic events and varied skill proficiencies of employees is studied. According to the experience of software management experts, actual dynamic events include the leave and return of employees, the arrival of new reworking tasks, and inaccurate estimations. Furthermore, the skill proficiencies of an employee change due to the learning and forgetting effect over time. Taking four objectives, the robustness and stability of a nondominated spark, the duration, and cost of a scheduling scheme, into account, the dynamic SPSP model is built. To solve this problem, an improved multiobjective firework algorithm with the following three contributions is proposed: (1) a spark or firework is encoded by two parts, the task allocation and the employee assignment, and corresponds to a scheduling scheme; (2) the explosion radius and the number of produced sparks for the explosion operator are adaptively adjusted in terms of the fireworks' status; and (3) mature and immature sparks are, respectively, restored to two archives. This reservation strategy avoids repeated searches in a small range and reduces the possibility of the algorithm quickly falling into local optimum. The experimental results show that the performance of IFA is the best compared with FA, NSGA-II, and MPSO. The robustness and stability of new objectives successfully guide IFA to obtain the more suitable nondominated solutions in each dynamic environment, which meet the need of software engineers. In addition, the dynamic characteristics of the employees' skills have significant impacts on the optimal schedule of a software project. In our future work, more unpredictable factors and complex situations, such as overtime pay and project outsourcing, shall be taken into account, and we will study more effective methods for solving dynamic software project scheduling problems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

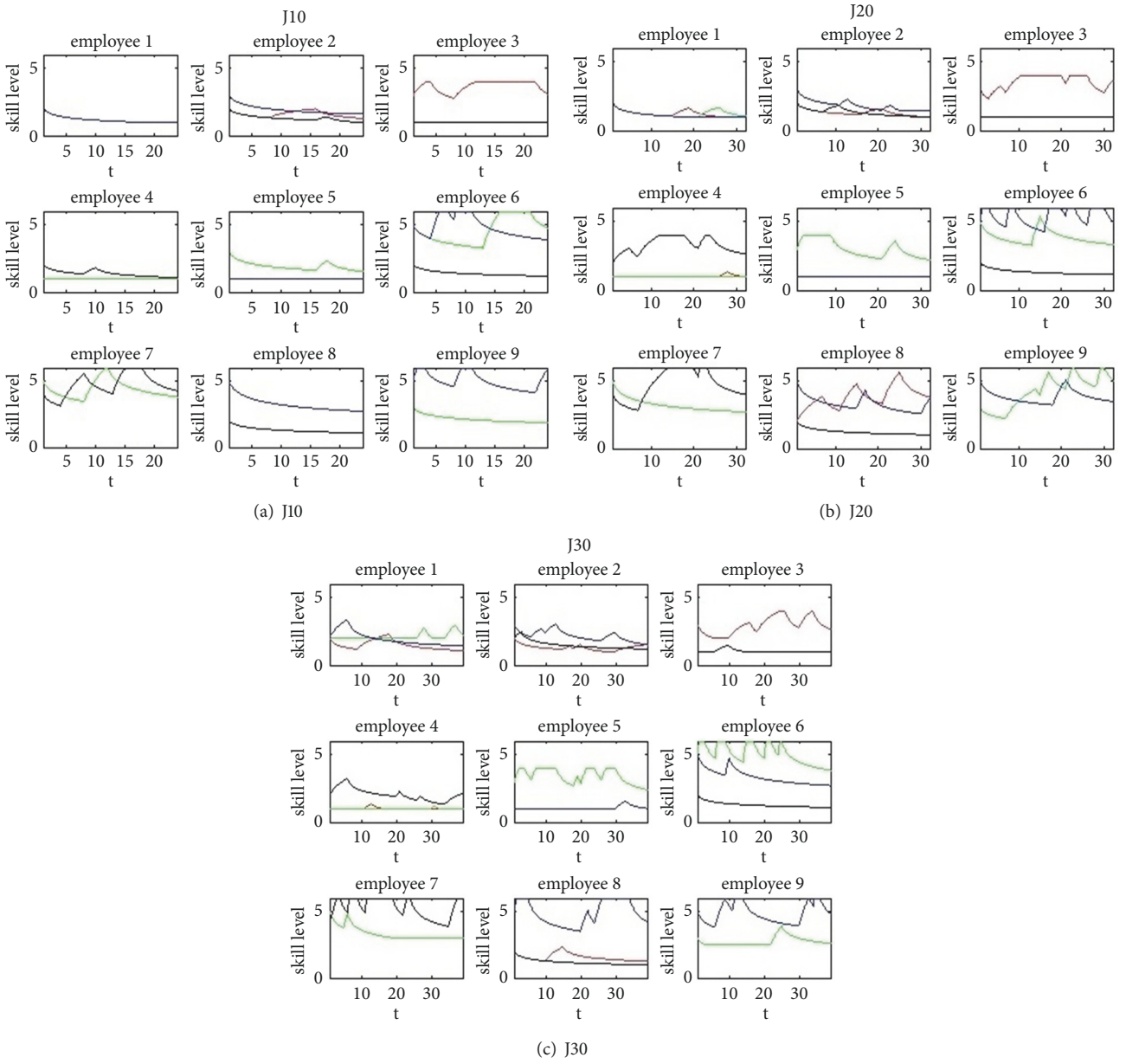


Figure 8: The curve of the skill proficiencies changing over time.

Acknowledgments

This work was jointly supported by the National Natural Science Foundation of China under Grant no. 61573361, Six Talent Peak Project in Jiangsu Province under Grant no. 2017-DZXX-046, National Key Research and Development Program under Grant no. 2016YFC0801406, and the State Key Laboratory of Robotics under Grant no. 2019-O21.

References

[1] M. André, M. G. Baldoquín, and S. T. Acuña, "Formal model for assigning human resources to teams in software projects," *Information and Software Technology*, vol. 53, no. 3, pp. 259–275, 2011.

[2] J. Duggan, J. Byrne, and G. J. Lyons, "A task allocation optimizer for software construction," *IEEE Software*, vol. 21, no. 3, pp. 76–82, 2004.

[3] E. Gleisberg, H. Zondag, and M. R. Chaudron, "An empirical study into the state of practice and challenges in IT project portfolio management," in *Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications (SEAA)*, pp. 248–257, Parma, Italy, September 2008.

[4] X. Shen, L. L. Minku, R. Bahsoon, and X. Yao, "Dynamic software project scheduling through a proactive-rescheduling method," *IEEE Transactions on Software Engineering*, vol. 42, no. 7, pp. 658–686, 2016.

[5] L. Wallace, M. Keil, and A. Rai, "Understanding software project risk: a cluster analysis," *Information and Management*, vol. 42, no. 1, pp. 115–125, 2004.

[6] A. Barreto, M. D. O. Barros, and C. M. L. Werner, "Staffing a software project: a constraint satisfaction and optimization-based approach," *Computers & Operations Research*, vol. 35, no. 10, pp. 3073–3089, 2008.

[7] E. Paiva, D. Barbosa, R. Lima, and A. Albuquerque, "Factors that influence the productivity of software developers in a developer view," in *Proceedings of the 2009 International Conference on Systems, Computing Sciences and Software Engineering, SCSS 2009, Part of the International Joint Conference on Computer, Information, and Systems Sciences, and Engineering, CISSE 2009*, pp. 99–104, 2010.

[8] K. Kang and J. Hahn, "Learning. forgetting curves in software development: does type of knowledge matter?" in *Proceedings of the International Conference on Information Systems*, vol. 194, Phoenix, Ariz, Usa, December 2009.

[9] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, vol. 8, McGraw-Hill, 2001.

[10] W. F. Boh and J. A. Espinosa, "Learning from experience in software sevelopment: s multilevel analysis," *Informs*, pp. 1315–1331, 2007.

[11] M. Davari and E. Demeulemeester, "Proactive-reactive resource-constrained project scheduling: a recovery-robust approach," in *Proceedings of the Business Analytics and Optimisation Conference*, Vienna, Austria, 2017.

[12] Y. Ge, "Software project rescheduling with genetic algorithms," in *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence, AICI 2009*, pp. 439–443, China, 2009.

[13] J. Xiao, L. J. Osterweil, Q. Wang, and M. Li, "Dynamic Resource Scheduling in Disruption-Prone Software Development Environments," in *Fundamental Approaches to Software Engineering*, vol. 6013 of *Lecture Notes in Computer Science*, pp. 107–122, Springer Berlin Heidelberg, Heidelberg, Germany, 2010.

[14] J. Xiao, L. J. Osterweil, Q. Wang, and M. Li, "Disruption-driven resource rescheduling in software development processes," in *New Modeling Concepts for Today's Software Processes*, vol. 6195 of *Lecture Notes in Computer Science*, pp. 234–247, Springer Berlin Heidelberg, Heidelberg, Germany, 2010.

[15] X. Shen, L. L. Minku, N. Marturi, Y. Guo, and Y. Han, "A Q-learning-based memetic algorithm for multi-objective dynamic software project scheduling," *Information Sciences*, vol. 428, pp. 1–29, 2018.

[16] Y. Guo, J. Ji, J. Ji, D. Gong, J. Cheng, and X. Shen, "Firework-based software project scheduling method considering the learning and forgetting effect," *Soft Computing*, pp. 1–16, 2018.

[17] B. W. Boehm, "Software Engineering Economics," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 1, pp. 4–21, 1984.

[18] U. Dorndorf, E. Pesch, and T. Phan-Huy, "A branch-and-bound algorithm for the resource-constrained project scheduling problem," *Mathematical Methods of Operations Research*, vol. 52, no. 3, pp. 413–439, 2000.

[19] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010.

[20] T. P. Wright, "Factors affecting the cost of airplanes," *Journal of the Aeronautical Sciences*, vol. 4, pp. 122–128, 1936.

[21] J. G. Carlson and A. J. Rowe, "How Much Does Forgetting Cost?" 1976.

[22] G. Li and S. Rajagopalan, "The impact of quality on learning," *Journal of Operations Management*, vol. 15, no. 3, pp. 181–191, 1997.

[23] F. Chen and C. Lee, "Minimizing the makespan in a two-machine cross-docking flow shop problem," *European Journal of Operational Research*, vol. 193, no. 1, pp. 59–72, 2009.

[24] Yi. Xiaoping, *Research on Employee Scheduling Problem Based on Learning Forgetting Effec*, Xidian University, 2014.

[25] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," *Advances in Swarm Intelligence*, pp. 355–364, 2010.

[26] Y. Guo, J. Cheng, S. Luo, D. Gong, and Y. Xue, "Robust dynamic multi-objective vehicle routing optimization method," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 6, pp. 1891–1903, 2018.

[27] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proceedings of the International Conference on Parallel Problem Solving for Nature*, vol. 1917, pp. 849–858, Springer Berlin, Heidelberg, Germany, 2000.

[28] E. Alba and J. Franciscochicano, "Software project management with GAs," *Information Sciences*, vol. 177, no. 11, pp. 2380–2401, 2007.

[29] Y.-n. Guo, H. Yang, M. Chen et al., "Ensemble prediction-based dynamic robust multi-objective optimization methods," *Swarm and Evolutionary Computation*, vol. 48, pp. 156–171, 2019.

[30] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[31] D. A. Van, V. Gary, and B. Lamont, "Multiobjective evolutionary algorithm research: a history and analysis," *Evolutionary Computation*, vol. 8, pp. 125–147, 1998.