Associate Professor Hongbo Li, PhD
E-mail: ishongboli@gmail.com, hongbo_li@t.shu.edu.cn
School of Management, Shanghai University, China

Hanyu Zhu, Master's Degree
E-mail: zhuhanyu1101@qq.com
School of Management, Shanghai University, China

Linwen Zheng, Master student
E-mail: linwenzheng@126.com (Corresponding author)
School of Management, Shanghai University, China

Associate Professor Yinbin Liu, PhD
E-mail: yinbinliu@126.com (Corresponding author)
School of Management, Shanghai University, China

SOFTWARE PROJECT SCHEDULING WITH MULTITASKING

Abstract*: In software development projects, employees tend to switch between various tasks within a time period. In addition, the task duration usually depends on the number of the allocated employees and their skill levels. We investigate the software project scheduling problem with multitasking and variable durations (SPSPM). We present a non-linear optimisation model that is then linearised into an equivalent mixed-integer linear programming model. To efficiently solve the NP-hard SPSPM, we design a two-stage priority rule-based heuristic algorithm and an improved genetic algorithm (GA). Extensive computational experiments are conducted on a benchmark dataset consisting of 540 project instances. The parameter settings of the GA are determined based on the Taguchi method for the Design of Experiment. The computational results obtained by comparing our algorithms with the exact solver CPLEX reveal that our GA is effective and competitive.*

Keywords*: Software project; Multitasking; Project scheduling; Integer programming; Meta-heuristic*

JEL Classification: M11, C44, C61

1. Introduction

The advancement of emerging information technologies such as artificial intelligence has provided new development opportunities for the software industry. In the meantime, a large number of software projects still suffer from low success rates. The CHAOS Report shows that only 29% of the investigated projects can be delivered on time within budget and customer requirements (Standish group, 2015).

DOI: 10.24818/18423264/57.1.23.10

The cause of the failures of many software development projects can be traced to the lack of effective scheduling (PMI, 2016). Software project scheduling aims to determine the start time of each task and who performs which task on the premise of satisfying task precedence relations, skill requirements, and resources availability constraints, such that a reasonable match between the employees with certain skills and tasks that require these skills forms, thereby minimising the project performance (e.g., project makespan, cost, etc.).

The existing studies on software project scheduling can be classified into two groups. The studies in the first group rely on mathematical optimisation models. Kazemipoor et al. (2013) study the integer programming and goal programming models for the multi-skill IT project scheduling problem, respectively. Huang et al. (2009) adopted the linear programming model for the personnel allocation problem. For the integrated optimisation problem of personnel assignment and scheduling, researchers have studied the integer programming model (Maenhout & Vanhoucke, 2017; Kolisch & Heimerl, 2012) and constraint programming (Hauder et al., 2020). Li & Womer (2009) propose an integer programming model and a Benders decomposition algorithm for software project scheduling. However, in the group of studies, it is usually assumed that the duration of the task is fixed and is not dependent on the number and skill levels of the personnel assigned to the tasks. Such assumptions are relatively strict, because the different number of employees assigned to the task and the different skill levels of the employees lead to differences in the duration of the task, which in turn will affect the cost of the project. In addition, the group of studies usually ignore multitasking, i.e., they assume that an employee can only engage in one task within a period of time. But in real-life software development, multitasking is not uncommon. For example, employees may participate in routine tasks such as progress reporting and code review every day, as well as other professional tasks (such as code writing, database design, etc.).

The other group of studies overcomes the above shortcomings at the cost of not explicitly presenting rigorous mathematical optimisation models. The representative research in this group comes from Alba & Chicano (2007). Based on the idea of Alba & Chicano (2007), many researchers have done a series of extended research (Xiao et al., 2013; Crawford et al., 2014; Minku et al., 2014; Luna et al., 2014), to make the research problem closer to the actual characteristics of the software project by considering the attributes of personnel wages, working days and holidays, skill proficiency and efficiency, and different types of human resources such as full-time and part-time. The algorithms involved in these studies are mainly meta-heuristic algorithms such as genetic algorithm (GA) (Minku et al., 2014; Li et al., 2018), ant colony algorithm (Crawford et al., 2014), multi-objective GA (Luna et al., 2014) and so on. However, on the one hand, in this group of studies, it is difficult to guarantee the optimality of the heuristic solutions and evaluate the effectiveness of the heuristic algorithms without mathematical optimisation models. On the other hand, this group of studies tends not to explicitly consider resource constraints; Instead, they usually treat the workload exceeding

the limits of employees as overtime, which is over-simplified and can lead to excessive overtime.

Therefore, this paper aims to fill the gap that the existing studies do not formulate and solve rigorous mathematical optimisation models for the software project scheduling problem with multitasking and variable durations. The main contributions of this paper are as follows:

(1) We propose the software project scheduling problem with multitasking (SPSPM) by considering various factors in software project management practice, such as allowing an employee to handle multiple tasks with different skills within a time period, the duration of a task depends on the skill characteristics of the employees assigned to it and resource availability constraints. (2) We present a nonlinear optimisation model for the SPSPM and devise a linearisation method to obtain its equivalent mixed-integer linear programming model. (3) Because the SPSPM is NP-hard, to solve large-scale SPSPM instances efficiently, we design a two-stage priority rule-based heuristic algorithm and an improved GA. We propose a specially designed encoding and decoding method by considering the characteristics of the SPSPM. (4) Extensive computational experiments are conducted on a benchmark dataset consisting of 540 instances. The parameter settings of the GA are determined based on the Taguchi method for the Design of Experiment (DOE). The performance of our algorithms is analysed by comparing them with the exact solver CPLEX.

2. Problem statement

The SPSPM is described as follows. We use a directed acyclic graph $G = (V, E)$ to represent a software project, in which the node set V denotes tasks in the project. The tasks are numbered from 0 to I, $V = \{0,1, \dots, I\}$. Tasks 0 and I are dummy tasks that denote the start and the end of the project, respectively. The duration of the dummy tasks is 0 and no resources are consumed when executing them. The directed arc set E denotes the precedence relationships between tasks, $E \subseteq V \times V$. If $(i, j) \in E$, there is a precedence relationship between tasks i and j, and task i is the predecessor of task j. Task j can only start after all its predecessor tasks have been completed.

The set of required skills during project execution is denoted as K. The set of the skills needed by task i is marked as K^i. When executing task i, the required workload related to skill k is WL_{ik} man-days, $WL_{ik} > 0$, $k \in K^i$. If $k \notin K^i$, then $WL_{ik} = 0$.

The employee set is denoted as R. Each employee masters one or more skills. The set of employees equipped with skill k is R^k, which means that there are $g_k = |R^k|$ employees who are able to use skill k. The binary parameter h_{rk} indicates whether employee $r \in R$ has the skill k,

$$h_{rk} = \begin{cases} 1 & if\ r \in R^k \\ 0 & otherwise \end{cases} \quad (1)$$

Every employee can deal with multiple tasks on a working day. θ_{rk} is the maximum available workload for employee $r \in R^k$ allocated to skill k per day. Without loss of generality, we standardise θ_{rk} as a decimal in the interval $[0,1]$. If employee r does not master skill l, $\theta_{rl} = 0$. For each employee, the overall workload in a day should not exceed the limit 1.

Employees need to be assigned tasks. Specifically, for each skill k involved in task i, at least one employee is required. To finish the task on time, the workload for employee r devoted to task i should not be less than the threshold $\gamma_{ik} \in [0,1]$, i.e., $\theta_{rk} \geq \gamma_{ik}$. The number of employees allocated to a task using skill k should not exceed g_k. In addition, the allocated employees are not allowed to be replaced after assignment.

The start and finish times of task i are s_i and c_i, respectively. The finish time c_I of the end dummy task equals to the makespan of the project. When there are multiple skills involved in task i, the earliest start (latest finish) time of the required skill by task i is the start time (finish time) of task i. We use s_{ik} and c_{ik} to indicate the start and end time of executing skill k in task i, respectively. The duration d_{ik} of each skill k in task i is variable, which is dependent on the working conditions of the assigned employees. The more the workload of employees finish in a day, the shorter the duration of the related task, i.e., $d_{ik} = WL_{ik} / \sum_{r \in R_{ik}} \theta_{rk}$, where R_{ik} is the set of employees using skill k allocated to task i. The deadline for the project is T_{max}.

The objective of the SPSPM is to minimise the human resource cost by generating a schedule that specifies what skills each employee should use to perform which tasks, as well as the start and finish time of each task, while satisfying the skill-task matching, precedence relationships, and the project deadline constraints. The project human resource cost consists of two parts, i.e., the fixed and the performance salary. The fixed salary is paid to employees as long as the project has not been completed. The total fixed salary is calculated as $\sum_{r \in R} C_r c_I$, where C_r is the daily fixed salary for employee r. The performance salary is paid to an employee only when she/he participates in a non-dummy task. The total performance salary of the project is calculated as $\sum_{r \in R} C'_r f_r$, where C'_r is the daily performance salary for employee r, and f_r is the number of days the employee actually works on the project.

3. Models

We first present a nonlinear programming model for the SPSPM. Then we devise a linearisation method to linearise the nonlinear model such that we can use exact solver (e.g., CPLEX) to solve the model.

3.1 Non-linear programming model

In addition to the time-related decision variables s_i, c_i, s_{ik}, c_{ik}, d_{ik} mentioned earlier, we also introduce three binary assignment variables x_{rikt}, y_{rik} and z_{rt}. Specifically, if employee r uses skill k to execute task i on the t-th day,

$x_{rikt} = 1$; otherwise $x_{rikt} = 0$. If employee r uses skill k to execute task i, $y_{rik} = 1$; otherwise $y_{rik} = 0$. If employee r is assigned to any non-dummy task on the t-th day, $z_{rt} = 1$; otherwise $z_{rt} = 0$. There is redundancy between these three variables, so we apply several extra logical restrictions on them: (a) For x_{rikt} and y_{rik}, if $y_{rik} = 0$, which means that employee r is not allocated to task i, then for any t, $x_{rikt} = 0$; if $x_{rikt} = 1$ on day t, which represents that employee r has been assigned to task i, then $y_{rik} = 1$. (b) For x_{rikt} and z_{rt}, similar restrictions are introduced. In addition, another role for x_{rikt} and z_{rt} is to associate the assignment variables with the time-related variables.

Based on the above descriptions, we formulate a nonlinear programming model (NLP) for the SPSPM:

(NLP) Minimise $\sum_{r\in R} C'_r f_r + C_r c_I$ (2)

Subject to:

$s_j \geq c_i$ $\forall (i,j) \in E \; and \; i \neq 0$ (3)

$s_i = \min s_{ik}$ $\forall i \in V, k \in K$ (4)

$c_i = \max c_{ik}$ $\forall i \in V, k \in K$ (5)

$s_{ik} + d_{ik} = c_{ik}$ $\forall i \in V, k \in K$ (6)

$c_I \leq \mathrm{T}_{max}$ (7)

$d_{ik} \geq WL_{ik} / \sum_{r\in R} \theta_{rk}\, y_{rik}$ $\forall i \in V, k \in K^i$ (8)

$\theta_{rk} \geq \gamma_{ik} y_{rik}$ $\forall r \in R, i \in V, k \in K^i$ (9)

$\sum_{i\in V} \sum_{k\in K} x_{rikt} \theta_{rk} \leq 1$ $\forall r \in R, t \in T$ (10)

$y_{rik} \leq h_{rk}$ $\forall r \in R, i \in V, k \in K^i$ (11)

$\sum_{r\in R} x_{rikt} \leq g_k$ $\forall i \in V, k \in \mathrm{K}, t \in T$ (12)

$\sum_{t\in T} x_{rikt} \geq d_{ik} - M(1 - y_{rik})$ $\forall r \in R, i \in V, k \in K^i$ (13)

$\sum_{t\in T} x_{rikt} \leq d_{ik} + M(1 - y_{rik})$ $\forall r \in R, i \in V, k \in K^i$ (14)

$\sum_{t\in T} z_{rt} = f_r$ $\forall r \in R$ (15)

$y_{rik} \leq \sum_{t\in T} x_{rikt}$ $\forall r \in R, i \in V, k \in K^i$ (16)

$M \cdot y_{rik} \geq \sum_{t\in T} x_{rikt}$ $\forall r \in R, i \in V, k \in K^i$ (17)

$z_{rt} \leq \sum_{i\in V} \sum_{k\in K} x_{rikt}$ $\forall r \in R, t \in T$ (18)

$M \cdot z_{rt} \geq \sum_{i\in V} \sum_{k\in K} x_{rikt}$ $\forall r \in R, t \in T$ (19)

$0 \leq s_i, c_i \leq \mathrm{T}_{max}, s_i, c_i \in Z$ $\forall i \in V$ (20)

$0 \leq s_{ik}, c_{ik} \leq \mathrm{T}_{max}, s_{ik}, c_{ik}, d_{ik} \in Z$ $\forall i \in V, k \in K$ (21)

$x_{rikt}, y_{rik}, z_{rt} \in \{0,1\}$ $\forall r \in R, i \in V, k \in K^i, t \in T$ (22)

$f_r \in Z^+$ $\forall r \in R$ (23)

In the NLP model, the objective function (2) minimises the total human resource cost of the project. Because the prolongation of the project makespan would increase the fixed cost, minimising function (2) can also indirectly achieve the objective of minimising the project makespan.

The constraints of the NLP model can be divided into three groups. The first group (constraints (3)-(7)) contains time-related constraints. Constraints in (3) describe the precedence relationships. In constraints (4) and (5), the start (finish) time of a task is determined by the earliest (latest) time of the assigned employees

when executing the task. Constraints in (6) indicate the relationship between the duration of a task and its start/finish time. Constraint (7) is the project deadline constraint.

The second group (constraints (8)-(15)) of constraints is related to assignment. Constraints (8) restrict the total workload devoted by the assigned employees should not be less than the required workload WL_{ik} of the task, which also influences the duration d_{ik}. Note that although the inequality sign is used in constraints (8), the inequality sign can also achieve the purpose to minimise d_{ik}, because the objective function also indirectly minimises the project makespan. Constraints (9) are the minimum workload constraints. The workload of an assigned employee r should not be less than the threshold γ_{ik}. Constraints (10) are the maximum workload constraints. The total workload per person per day cannot exceed 1. Constraints (11) mean that employees are not able to use the skills that they do not have. Constraints (12) represent the limits of the number of employees. Constraints (13) and (14) guarantee that the employees are not be replaced after assignment, and M is a sufficiently large positive integer. Constraints (15) are used to calculate the number of actual working days of employee r.

The third group (constraints (16)-(19)) of constraints describe the logical relationships among the assignment variables. Constraints (16) and (17) reflect the logical relationships between x_{rikt} and y_{rik}. Constraints (18) and (19) describe the relationships between x_{rikt} and z_{rt}. Finally, constraints (20)-(23) define the range of the decision variables. In addition, it can be seen that the constraints (4), (5) and (8) are nonlinear.

3.2 Model linearisation

In this subsection, we linearise the model NLP. For the nonlinear constraints, the constraints in (8) are the most difficult to linearise. We tackle this problem by replacing d_{ik} in the constraint with auxiliary variables to form a linear constraint.

First, we sum the daily workload devoted by employees using skills in a task, and obtain the equivalent constraints of (8):

$$\sum_{r\in R}\sum_{t\in T} x_{rikt}\cdot\theta_{rk} \geq WL_{ik} \qquad \forall i\in V, k\in K^i \tag{24}$$

Then, to replace d_{ik} in constraints (6), (13) and (14), we introduce a binary auxiliary variable e_{ikt}. If skill k of task i is executed on the t-th day, $e_{ikt}=1$; otherwise $e_{ikt}=0$. According to the above definition, there are logical relationships between e_{ikt} and x_{rikt}: if $e_{ikt}=0$, which means that task i is not executed by any employee with skill k on the t-th day, then for any employees r, $x_{rikt}=0$. If for a day t, $x_{rikt}=1$, which means that employee r is assigned to task i, then $e_{ikt}=1$. These logical restrictions can be quantified as follows:

$$e_{ikt} \leq \sum_{r\in R} x_{rikt} \qquad \forall i\in V, k\in K^i, t\in T \tag{25}$$

$$M\cdot e_{ikt} \geq \sum_{r\in R} x_{rikt} \qquad \forall i\in V, k\in K^i, t\in T \tag{26}$$

Actually, e_{ikt} converts the original d_{ik} into a set of binary variables:

$$d_{ik} = \sum_{t\in T} e_{ikt} \qquad \forall i\in V, k\in K^i \tag{27}$$

According to (27), constraints (6), (13) and (14) can be transformed into a new set of equivalent linear constraints:

$$\sum_{t\in T} e_{ikt} = c_{ik} - s_{ik} + 1 \qquad \forall i \in V, k \in K^i \qquad (28)$$

$$\sum_{t\in T} x_{rikt} \geq \sum_{t\in T} e_{ikt} - M(1 - y_{rik}) \qquad \forall r \in R, i \in V, k \in K^i \qquad (29)$$

$$\sum_{t\in T} x_{rikt} \leq \sum_{t\in T} e_{ikt} + M(1 - y_{rik}) \qquad \forall r \in R, i \in V, k \in K^i \qquad (30)$$

Finally, the following auxiliary constraints ensure that there are no interruptions during project execution:

$$s_{ik} \leq t \cdot e_{ikt} + M(1 - e_{ikt}) \qquad \forall i \in V, k \in K^i, t \in T \qquad (31)$$

$$c_{ik} \geq t \cdot e_{ikt} \qquad \forall i \in V, k \in K^i, t \in T \qquad (32)$$

Based on the above linearisation method, we obtain the mixed-integer linear programming model (MILP) that is equivalent to model NLP:

(MILP) Minimise (2)

Subject to:

(3), (7), (9)-(12), (15)-(23), (24)-(32)

$$s_i \leq s_{ik} \qquad \forall i \in V, k \in K \qquad (33a)$$

$$s_i \geq s_{ik} - M \cdot u_{ik} \qquad \forall i \in V, k \in K \qquad (33b)$$

$$\sum_{k\in K^i} u_{ik} = |K| - 1 \qquad \forall i \in V \qquad (33c)$$

$$u_{ik} \in \{0,1\} \qquad \forall i \in \mathrm{V}, k \in K \qquad (33d)$$

$$c_i \geq c_{ik} \qquad \forall i \in V, k \in K \qquad (34a)$$

$$c_i \leq c_{ik} + M \cdot v_{ik} \qquad \forall i \in V, k \in K \qquad (34b)$$

$$\sum_{k\in K^i} v_{ik} = |K| - 1 \qquad \forall i \in V \qquad (34c)$$

$$v_{ik} \in \{0,1\} \qquad \forall i \in \mathrm{V}, k \in K \qquad (34d)$$

where the linear constraints (33) and (34) correspond to constraints (4) and (5), and u_{ik}, v_{ik} $(i \in \mathrm{V}, k \in K)$ are binary auxiliary variables.

Proposition 1. The SPSPM is NP-hard.

Proof. Consider a special case of the SPSPM: The threshold of workload is ignored, i.e., γ_{ik} are set to 0, and the fixed salary for every employee is set to $1/|\mathrm{R}|$. This special case has been studied by Xiao et al. (2013) and proved to be NP-hard. Therefore, as a generalisation, the SPSPM is NP-hard. ■

4. Algorithms

Since the SPSPM is NP-hard, to solve the large-scale SPSPM instances efficiently, we design a two-stage priority rule-based heuristic algorithm and an improved GA in this section.

4.1 Two-stage priority rule-based heuristic algorithm

The pseudocode of our two-stage priority rule-based heuristic algorithm is shown in Algorithm 1. In Algorithm 1, tasks are first scheduled, and then employees are assigned. Specifically, (a) the scheduling order of tasks using the minimum total workload priority rule (Browning & Yassine, 2010); (b) the assignment order of employees is determined according to the rule that the maximum committable workload of the employees for a certain skill is from highest to lowest. Finally, the start time and assignment for each task can be obtained.

Algorithm 1. Two-stage priority rule-based heuristic algorithm

Step 1: Initialisation.

Add dummy task 0 to the completed task set C_g. The rest tasks are added to the unschedulable task set S_g. The schedulable task set D_g is set to empty.

Step 2: Select a task to schedule.

If D_g is empty and the number of tasks in C_g is less than the total number of tasks, then the tasks whose predecessor tasks have been completed are chosen from S_g and appended into D_g.

If D_g is not empty, then the task with the minimum workload is selected from D_g and denoted as i^*.

Step 3: Assign employees for the selected task.

Step 3.1: For each skill k^* involved in task i^*, calculate the total amount of workload that all employees can put in it, i.e., $eff\_sum_{i^*k^*} = \sum_{r \in R} \theta_{rk^*}, \theta_{rk^*} \geq \gamma_{i^*k^*}$.

Step 3.2: For the skill, assign employees with the maximum $eff\_sum_{i^*k^*}$.

Step 4: Determine the start time $s_{i^*k^*}$ of skill k^* in task i^*.

Step 4.1: Assign all available employees with skill k^* to the task i^*, $d_{i^*k^*} = \lceil WL_{i^*k^*}/eff_{sum_{i^*k^*}} \rceil$

Step 4.2: The start time $s_{i^*k^*}$ of executing task i^* with skill k^* is not supposed to be earlier than the finish time $c_{i^*}^{pre}$ of i^*'s predecessor tasks. And the assigned employees should not be occupied by other tasks (the earliest available time for the assigned employees is denoted as $t_{i^*k^*}^{available}$). Therefore, the start time for a skill to be used is $s_{i^*k^*} = \max\{c_{i^*}^{pre}, t_{i^*k^*}^{available}\}$, the finish time is $c_{i^*k^*} = s_{i^*k^*} + d_{i^*k^*} - 1$.

Step 5: Decide the start time s_{i^*} of task i^*.

Repeat steps 3 and 4 until all skills in task i^* are assigned to employees. Then, move **task** i^* from D_g to C_g. For task i^*, $s_{i^*}(c_{i^*})$ is obtained according to **Equations (4)-(5)**.

Step 6: Repeat steps 2 to 5 until the number of tasks in C_g equals the total number of tasks.

4.2 Improved genetic algorithm

To solve the SPSPM more effectively, we also develop an improved GA. GA is a classical population-based meta-heuristic algorithm, which has been widely applied in project scheduling (Alba & Chicano, 2007; Li et al., 2018).

4.2.1 Encoding of a schedule

A schedule is encoded into a chromosome $ch = \{RK, RL, SL, DM\}$, where (a) the random key vector $RK = (a_0, a_1, \dots, a_i, \dots, a_I)$ is used to indicate the order of tasks to be scheduled, where $a_i \in [0,1]$ represents the priority value of task i. (b) The employee vector $RL = (r_0, r_1, \dots, r_i, \dots, r_I)$ is used to indicate the order of employees assigned to each task, where r_i is an employee list for task i, and there are $|K|$ sub-lists in r_i, the k-th sub-list is the order of employees to be assigned for skill k ($k = 1,2, \dots, K$). If a skill is not involved in the task, the corresponding sub-list is empty. (c) In the skill vector $SL = (SK_0, SK_1, \dots, SK_i, \dots, SK_I)$, $SK_i = (sk_1, sk_2, \dots, sk_{K^i})$ is the order of $|K^i|$ skills involved in task i. (d) The employee demand vector $DM = (dm_0, dm_1, \dots, dm_i, \dots, dm_I)$ is used to show the number of employees assigned to task i, where $dm_i = (dm_i d_1, dm_i d_2, \dots, dm_i d_{|K|})$ is a $|K|$-dimension vector and the value of each element is a decimal between 0 and 1 indicating the proportion of the number of employees assigned to a skill in task i to the total number of available employees.

4.2.2 Decoding

We design a decoding procedure to transform a chromosome into a schedule (Algorithm 2). In Algorithm 2, we first initialise the parameters. Next, at the start of each iteration, we update D_L. If D_L is not empty, for each task i^* in D_L, we obtain the arrangement list (SkillList_{i^*}). Then, we choose available *resource_need* employees according to the preferred employee priority assignment method (Algorithm 3) (Line 6-11). After that, we update $WL_{i^*k^*}$ based on the workload completed by the assigned employees ($R_{i^*\text{k}^*}^{ass}$) (Line 11). When $WL_{i^*\text{k}^*} \leq 0$, the demand for skill k^* in task i^* has been satisfied (Line 12). If all the requirements of skills in task i are satisfied, we update D_L and C_L (Lines 15-16).

Algorithm 2. Decoding procedure

Input: Project parameters, RK, RL, SL, DM
Output: $x_{rikt}, \text{s}_{ik}, \text{c}_{ik}$

1 Initialise: $C_L, D_L, S_L, x_{rikt}, s_{ik}, c_{ik}, t \leftarrow 0$
2 While $t < \text{T}_{\max}$:
3 If $\text{len}(D_L) == 0$:
4 From S_L, select tasks whose predecessors have been added to C_L, and add them to D_L in the order of S_L
5 Else
6 For i^* in D_L:
7 $\text{SkillList}_{i^*} \leftarrow SK_{i^*}$ in SL
8 For k^* in SkillList_{i^*}:
9 $resource\_need \leftarrow ceil(dm_{i^*} d_{k^*} * |R_{i^*k^*}|)$, where $R_{i^*k^*} = \{r | \theta_{rk^*} \geq \gamma_{i^*k^*}, r \in R\}$. $\text{R\_List}_{i^*k^*} \leftarrow \text{r}_{i^*}[k^*]$ in RL
10 Based on Algorithm 3, select up to *resource_need* employees from $\text{R\_List}_{i^*k^*}$, update $\text{s}_{i^*k^*}$ and $x_{ri^*k^*t}$
11 Calculate the rest workload of $WL_{i^*\text{k}^*}$ after the t-th day, i.e.,$WL_{i^*k^*} \leftarrow WL_{i^*k^*} - \sum_{r \in R_{i^*\text{k}^*}^{ass}} \theta_{rk^*}, \sum_{r \in R_{i^*\text{k}^*}^{ass}} \theta_{rk^*}$
12 If $WL_{i^*\text{k}^*} \leq 0$: $\text{c}_{i^*k^*} \leftarrow \text{t}$ End if
13 End for
14 End for
15 Move the finished tasks in D_L to C_L
16 $t \leftarrow t + 1$
17 End if
18 End While

We use an example to explain Algorithm 3. As shown in Figure 1(a), the preferred employee for task A is $R1$, but $R1$ is occupied at t_0, thus employee $R0$ performs the task; at time t_1, $R1$ returns to the idle state again, and if more than 70% of the workload of task A has been completed, the reassignment of $R1$ to the task is abandoned (Figure 1(b)); otherwise, reassign $R1$ to participate in task A and restore the workload required for task A so that it starts at time t_1 (Figure 1(a)).

Algorithm 3. Preferred employee priority assignment method

```
1   For r in R_List_{i*k*}:
2     If resource_need >0:
3       If r is available in day t:
4         If day t is the first day to process skill k* of activity i*:
5           resource_need ← resource_need -1
6           x_{ri*k*t} ← 1
7           update the rest workload of skill k* of activity i* and available efficiency of resource r
    at day t
8           s_{i*k*} ← t
9         Else:
10          If the ranking of employee r is higher than that of assigned employee r':
11            If the rest workload of skill k* of activity i* is more than 30% of original workload:
12              Reset x_{r'i*k*t'} (r' ∈ R, t' ∈ [0, t − 1]) and restore the WL_{i*k*} to initial value
13              Repeat line 5 to 8
14            End if
15          Else:
16            Repeat line 5 to 7
17          End if
18        End if
19      End if
20    End if
21  End for
```

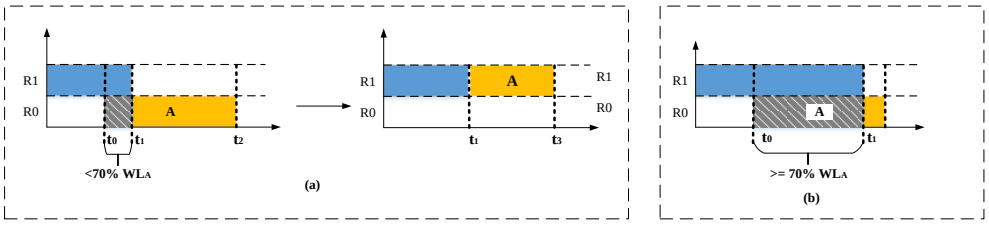

Figure 1. Improvement of the decoding procedure

4.2.3 Population initialisation

There are POP individuals in the population. In the initial population, the random key vectors are generated randomly. The employee vectors are generated based on two heuristic methods, and each heuristic method generate $POP/2$ individuals: (a) The employees are ranked according to the ascending order of their performance salary. (b) The employees are ranked in the descending order indicated by their cost performance (for skill k, the cost performance of employee r is the ratio of its performance salary C_r' to its maximum available workload θ_{rk}).

For the skill vectors, the skills in a task are ranked on the descending order of $eff_{sum_{ik}}(i \in V, k \in K^i)$. For the employee demand vectors, in the n-th ($n = 1,2 \ldots POP$) individual, all the non-zero random keys in DM are set to n/POP. It should be noted that if $k \in K^i$, $dm_i d_k \neq 0$.

4.2.4 Crossover, mutation and selection operators

We randomly pair the individuals in the population to form $POP/2$ pairs of parent individuals. For paired individuals, the crossover and mutation operators are applied to RK and DM; SL and RL are directly copied to their child individuals.

In crossover operator, a random number $r \in (0,1)$ is generated first. If $r \leq R_c$ (crossover probability), the average value of the values on RK and DM of the parent individuals is taken as the corresponding value of the child individual. Otherwise, the parent individuals will conduct an average calculation with two special chromosomes, of which all random key values on RK and DM are 1 or 0.

After crossover, for the child individuals, the probability of mutation of each gene is R_m. If a gene is selected to mutate, we randomly choose a position on RK and change its value to a newly generated random decimal between (0,1). Then, we choose a non-zero position on DM and change its value in the same way as above.

After crossover and mutation, we update the population. We first choose the best $R_e \times POP$ individuals from the parent population and copy them directly to the new population ch'', where R_e is the rate of the elite. Then, we obtain the rest individuals in ch'' by the following way: we sample $POP - R_e \times POP$ times. In each sampling process, two individuals are randomly selected from ch', and the individual with the lower objective function value is added to the population ch''.

5. Computational experiments

The proposed algorithms have been implemented in Python 3.7. Our computational experiments are conducted on a computer with Intel Core i5 3.20 GHz CPU and Windows 7 64-bit.

5.1 Benchmark dataset

Because there is no suitable dataset for our problem at present, we generate the benchmark dataset based on full-factorial experimental design. We use the project scheduling problem instance generator *RanGen* (Demeulemeester et al., 2003) to produce the project networks. There are two parameters in *RanGen* that define the structure of the network: the number of tasks and the order strength (OS). The OS is the ratio between the number of precedence relationships in the network and the theoretically maximum number of precedence relationships. The network with a higher OS value has a higher network density. We specify various levels for the number of activities, OS, the number of employees, and the number of skills (Table 1), and generate five instances for each parameter combination. Therefore, we obtain a total of $4 \times 3 \times 3 \times 3 \times 5 = 540$ project instances.

Table 1. Parameter settings in the benchmark dataset

Parameter	Values			
Number of tasks (I)	5	10	20	30
Order strength (OS)	0.3	0.5	0.7	
# of employees ($\|R\|$)	4	6	8	
# of skills ($\|K\|$)	3	5	7	

Table 2. Employee's average available workload and salaries

Average devoted workload	Daily fixed salary	Daily performance salary
$\theta_r \geq 0.75$	[31,40]	[61,70]
$0.75 > \theta_r \geq 0.5$	[21,30]	[51,60]
$\theta_r < 0.5$	[11,20]	[41,50]

The number of skills mastered by each employee is sampled from the discrete interval $[1, \lceil |K|/2 \rceil]$. The maximum devoted workload for employee θ_{rk} is chosen randomly from the set $\{0.25, 0.5, 0.75, 1.0\}$. The number of skills involved in each task is also sampled from the discrete interval $[1, \lceil |K|/2 \rceil]$. The workload WL_{ik} required by each skill is chosen from the set $\{3,4,5\}$. The threshold γ_{ik} of workload is calculated by $1/4$ multiplying a number chosen from the discrete interval $[1, 4 \times \theta^*_{rk}]$, where $\theta^*_{rk} = \max_r \theta_{rk}$. This approach ensures that there is at least an employee who meets the threshold requirements. For the fixed salary and performance salary of every employee, they are randomly chosen from different intervals shown in Table 2 based on the employee's average available workload $\theta_r = \sum_{k \in K^r} \theta_{rk} / |K^r|$, where K^r is the skill set mastered by employee r.

We use CPLEX 12.9 to solve the instances. The maximum time to solve an instance is limited to 600 seconds. The value of M in model MILP is set to 1×10^8. Based on the calculation results, we divide the benchmark dataset into 3 subsets (Table 3), i.e., SET_O contains instances that have been solved optimally, SET_F contains instances with only feasible solutions, and SET_I contains instances whose feasible solutions are not obtained.

Table 3. Number of instances in different subdatasets

# of tasks	SET_O	SET_F	SET_I
5	105	28	2
10	16	50	69
20	0	0	135
30	0	0	135
Sum	121	78	341

Table 4. Levels of parameter values

Levels	POP	R_c	R_m	R_e
1	50	0.5	0.15	0.04
2	100	0.7	0.35	0.08
3	200	0.9	0.55	0.12

5.2 Parameter settings

We applied the Taguchi method of DOE based on SET_O to determine the best values of parameters in the proposed GA. There are four parameters in the GA: the size of the population (POP), the probability of crossover (R_c), the probability of mutation (R_m) and the rate of elite (R_e). As shown in Table 4, we set 3 levels for each parameter.

The average response variable (ARV) is defined as follows:

$$\text{ARV} = \frac{\sum_{i \in \text{SET}_\text{O}} [(obj_i - opt_i)/opt_i]}{|\text{SET}_\text{O}|} \tag{35}$$

where opt_i is the optimal objective function value of the i-th instance obtained by CPLEX, obj_i is the objective value calculated by the GA. We choose the $L_9(3^4)$ orthogonal table, and the termination condition of the GA is to generate up to 1,000 schedules. The orthogonal array and values of ARV are shown in Table 5. Table 6 displays the range of the ARV values as well as the significant rank of each factor. Figure 2 describes the changing trend of the ARV. It can be seen that the parameter that has the most impact on the GA is the probability of mutation (R_m), the second one is the rate of elite (R_e), followed by the size of population (POP) and the probability of crossover (R_c). Therefore, the final parameter settings are POP= 50 (level 1), $R_c = 0.7$ (level 2), $R_m = 0.55$ (level 3) and $R_e = 0.12$ (level 3). These parameter settings will be adopted in the following experiments.

Table 5. Orthogonal array and values of ARV

No.	Factors				ARV
	POP	R_c	R_m	R_e	
1	1	1	1	1	0.0327
2	1	2	2	2	0.0275
3	1	3	3	3	0.0218
4	2	1	2	3	0.0297
5	2	2	3	1	0.0264
6	2	3	1	2	0.0306
7	3	1	3	2	0.0287
8	3	2	1	3	0.0329
9	3	3	2	1	0.0365

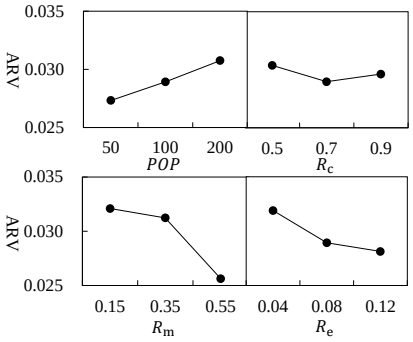

Figure 2. GA factor level trend

Table 6. Average ARV and rank for each factor

Levels	POP	R_c	R_m	R_e
1	**0.0273**	0.0304	0.0321	0.0319
2	0.0289	**0.0290**	0.0312	0.0289
3	0.0307	0.0296	**0.0256**	**0.0281**
Range	0.0034	0.0014	0.0065	0.0038
Rank	3	4	1	2

5.3 Main results

We apply our two-stage priority rule-based heuristic algorithm and GA to solve the benchmark dataset. For each instance, we use the GA to obtain three solutions and calculate the average objective function value as the final results of the GA. The termination condition of our GA is to generate up to 1,000 schedules.

For the instances in SET_O and SET_F, we use the average relative deviation $ARD(M,SET) = \frac{\sum_{i\in SET}[(obj_i^M - obj_i^{CPLEX})/obj_i^{CPLEX}]}{|SET|} \times 100\%,\ (M \in \{GA,\ HA\},\ SET \in \{SET_O,\ SET_F\})$ from the results obtained by CPLEX to evaluate the performance of our algorithms, where obj_i^{GA} and obj_i^{HA} are the values of the objective function of the i-th instance calculated by the GA and the two-stage priority rule-based heuristic algorithm respectively, and obj_i^{CPLEX} is the best objective value obtained by CPLEX. For the instances in SET_I, we compare the results of the GA with the priority rule-based heuristic algorithm, and the metric $Imp = \frac{\sum_{i\in SET_I}[(obj_i^{HA} - obj_i^{GA})/obj_i^{HA}]}{|SET_I|} \times 100\%$ is adopted to measure the improvement of the GA compared with the priority rule-based heuristic algorithm. The average CPU times to solve each instance are used to evaluate the efficiency of the algorithms.

Table 7 shows the results on SET_O. From Table 7, we can see that our GA performs well. The average gap to the optimal solutions is only 2.37%. In most cases, the CPU times of our GA are less than those of CPLEX. Our GA is far better than the priority rule-based heuristic algorithm. Table 8 displays the results on SET_F, from which we can see that our GA obtains better solutions than CPLEX (the average gap is negative, which means that our GA is better) and spends less CPU times. The average gap between the heuristic algorithm and CPLEX is 22.04%. Table 9 gives the results on SET_I, showing that the proposed GA is significantly better than the heuristic algorithm.

In summary, for the easy instances, our GA is able to get solutions that are very close to the optimal solutions in a short time. For the difficult instances, the performance of our GA is far better than CPLEX and our priority rule-based heuristic algorithm, and the GA is able to obtain satisfying solutions in a short time. In addition, the quality of the solution calculated by our GA is higher than that of our priority rule-based heuristic algorithm. The proposed GA is effective and efficient.

Table 7. The computational results on SET_O

Number of tasks	*OS*	ARD		CPU(s)	
		GA	HA	GA	CPLEX
5	0.3	2.19%	36.12%	90.41	64.95
	0.5	2.02%	30.54%	93.36	144.95
	0.7	1.73%	25.67%	93.43	141.78
10	0.3	4.37%	36.57%	95.03	365.68
	0.5	7.12%	51.16%	94.15	296.72
	0.7	3.99%	32.63%	98.43	277.32
Average		**2.37%**	**31.52%**	**93.11**	**145.99**

Table 8. The computational results on SET_F

Number of tasks	*OS*	ARD		CPU (s)	
		GA	HA	GA	CPLEX
5	0.3	-8.36%	25.40%	97.38	600.19
	0.5	-8.31%	6.91%	100.25	600.70
	0.7	-9.72%	11.90%	104.97	600.11
10	0.3	-5.19%	28.30%	95.11	600.83
	0.5	-4.57%	23.30%	97.69	600.19
	0.7	-8.83%	17.46%	103.16	600.09
Average		**-7.10%**	**22.04%**	**98.93**	**600.35**

Table 9. The computational results on SET_I

Number of tasks	*OS*	Average *Imp* of GA	CPU times (s) of GA
5	0.5	25.21%	101.16
	0.7	14.82%	107.81
10	0.3	27.43%	105.99
	0.5	25.16%	113.36
	0.7	22.42%	119.31
20	0.3	23.78%	125.78
	0.5	22.86%	136.28
	0.7	22.04%	161.31
30	0.3	20.97%	155.28
	0.5	22.14%	175.73
	0.7	19.47%	212.95
Average		**22.51%**	**151.10**

5.4 Sensitivity analysis

We further investigate the factors that influence the GA. First, from Tables 7-9, we can see that the OS value does not have a significant impact on the results, which means that the GA is robust for the different number of precedence relationships in a project.

Then, we consider the impact of the number of skills $|K|$ and the number of employees $|R|$ in different datasets (see Figure 3). Note that in Figure 3, some points/lines are not displayed because of the lack of corresponding combination instances in the dataset. We can find from Figure 3 that as the number of skills increases, the performance of the GA becomes worse, which is more significant in small-scale instances. For the number of employees, in the small-scale instances, the impact of the GA is weakened as $|R|$ increases. But in large-scale instances, this phenomenon is the opposite, which means that our GA is more suitable for large-scale software project scheduling problems.

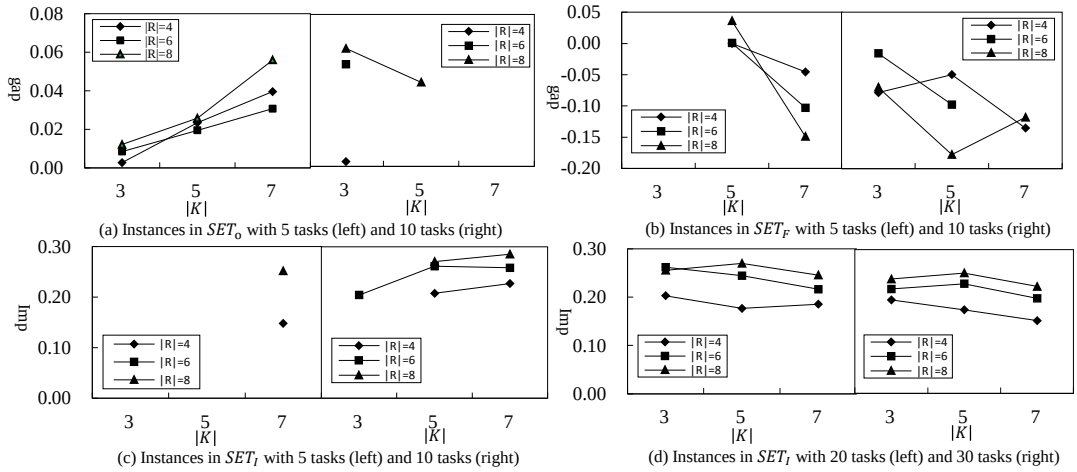

Figure 3. Impacts of different factors on the GA

6. Conclusions and future research

Taking into account multitasking and variable task durations in software project, we investigate the SPSPM. For this problem, we formulate a nonlinear programming model, which is then transformed into an equivalent mixed-integer linear programming model. To effectively solve the NP-Hard SPSPM, a two-stage priority rule-based heuristic algorithm and an improved GA are proposed. A benchmark dataset consisting of 540 project instances is generated by using full-factorial experimental design. Based on the larger-scale computational experiments, we analyse the performance of our proposed algorithms by comparing the them with CPLEX, which reveals the effectiveness and advantages of our GA. The sensitivity analysis also indicates that our GA is robust and that our GA is especially suitable for solving large-scale instances.

Future work will further develop more efficient algorithms for the SPSPM. It would also be interesting to consider more realistic factors faced by software project management, such as uncertainty (Li et al., 2020), resource leveling (Li et al., 2021), and preemption (Liu et al., 2019).

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant Number 71602106).

REFERENCES

[1] **Alba, E., Chicano, J.F. (2007),** ***Software project management with Gas***. *Information Sciences*, 177(11), 2380-2401;

[2] **Browning, T. R., Yassine, A. A. (2010),** ***Resource-constrained multi-project scheduling: Priority rule performance revisited***. *International Journal of Production Economics*, 126(2), 212-228;

[3] **Crawford, B., Soto, R., Johnson, F., Monfroy, E., Paredes, F. (2014),** ***A max-min ant system algorithm to solve the software project scheduling problem***. *Expert Systems with Applications,* 41(15), 6634-6645;

[4] **Demeulemeester, E., Vanhoucke, M., Herroelen, W. (2003),** ***RanGen: A random network generator for activity-on-the-node networks***. *Journal of scheduling*, 6(1), 17-38;

[5] **Hauder, V.A., Beham, A., Raggl, S., Parragh, S.N., Affenzeller, M. (2020),** ***Resource-constrained multi-project scheduling with activity and time flexibility*****.** *Computers & Industrial Engineering*, 150, 106857;

[6] **Huang, H.C., Lee, L.H., Song, H., Eck, B.T. (2009),** ***SimMan – A simulation model for workforce capacity planning.*** *Computers & Operations Research*, 36(8), 2490-2497;

[7] **Kazemipoor, H., Tavakkoli-Moghaddam, R., Shahnazari-Shahrezaei, P., Azaron, A. (2013),** ***A differential evolution algorithm to solve multi-skilled project portfolio scheduling problems***. *The International Journal of Advanced Manufacturing Technology*, 64(5), 1099-1111;

[8] **Kolisch R., Heimerl C. (2012),** ***An efficient metaheuristic for integrated scheduling and staffing it projects based on a generalized minimum cost flow network***. *Naval Research Logistics*, 59(2), 111-127;

[9] **Li, H., Hu, Z., Zhu, H., Liu, Y. (2021),** ***Preemptive resource leveling in projects*****.** *Economic Computation and Economic Cybernetics Studies and Research*, 55(4), 51-68;

[10] **Li H., Womer K. (2009),** ***Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm***. *Journal of Scheduling*, 12(3), 281;

[11] **Li, H., Xiong, L., Liu, Y., Li, H. (2018),** ***An effective genetic algorithm for the resource levelling problem with generalised precedence relations***. *International Journal of Production Research*, 56(5), 2054-2075;

[12] **Li, H., Zhang, X., Sun, J., Dong, X. (2020),** ***Dynamic resource levelling in projects under uncertainty***. *International Journal of Production Research*, 1-21;

[13] **Liu, Y., Hu, Z., Li, H., Zhu, H. (2019),** ***Does preemption lead to more leveled resource usage in projects? a computational study based on mixed-integer linear programming***. *Economic Computation and Economic Cybernetics Studies and Researc*h, 53(4), 243-258;

[14] **Luna F., González-álvarez, D. L., Chicano, F., Vega-Rodríguez, M.A. (2014),** ***The software project scheduling problem: a scalability analysis of multi-objective metaheuristics*****.** *Applied Soft Computing*, 15(2), 136-148.;

[15] **Maenhout, B., Vanhoucke, M. (2016),** ***An exact algorithm for an integrated project staffing problem with a homogeneous workforce*****.** *Journal of Scheduling*, 19(2), 107-133;

[16] **Minku L.L., Sudholt, D., Yao, X. (2014),** ***Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis***. *IEEE Transactions on Software Engineering*, 40(1), 83-102;

[17] **PMI (Project Management Institute). (2016)**, ***Pulse of the profession 2016: The high cost of low performance***;

[18] **Standish Group. (2015)**, ***CHAOS Report*****,** 2015, http://www.standishgroup.com;

[19] **Xiao, J., Ao, X. T., Tang, Y. (2013),** ***Solving software project scheduling problems with ant colony optimization.*** *Computers & Operations Research*, 40(1), 33-46.