Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler

Wei-Neng Chen, *Member*, *IEEE*, and Jun Zhang, *Senior Member*, *IEEE*

Abstract—Research into developing effective computer aided techniques for planning software projects is important and challenging for software engineering. Different from projects in other fields, software projects are people-intensive activities and their related resources are mainly human resources. Thus, an adequate model for software project planning has to deal with not only the problem of project task scheduling but also the problem of human resource allocation. But as both of these two problems are difficult, existing models either suffer from a very large search space or have to restrict the flexibility of human resource allocation to simplify the model. To develop a flexible and effective model for software project planning, this paper develops a novel approach with an event-based scheduler (EBS) and an ant colony optimization (ACO) algorithm. The proposed approach represents a plan by a task list and a planned employee allocation matrix. In this way, both the issues of task scheduling and employee allocation can be taken into account. In the EBS, the beginning time of the project, the time when resources are released from finished tasks, and the time when employees join or leave the project are regarded as events. The basic idea of the EBS is to adjust the allocation of employees at events and keep the allocation unchanged at nonevents. With this strategy, the proposed method enables the modeling of resource conflict and task preemption and preserves the flexibility in human resource allocation. To solve the planning problem, an ACO algorithm is further designed. Experimental results on 83 instances demonstrate that the proposed method is very promising.

Index Terms—Software project planning, project scheduling, resource allocation, workload assignment, ant colony optimization (ACO)

1 Introduction

With the rapid development of the software industry, software companies are now facing a highly competitive market. To succeed, companies have to make efficient project plans to reduce the cost of software construction [1]. However, in medium to large-scale projects, the problem of project planning is very complex and challenging [2]. In fact, in China it was reported that more than 40 percent of unsuccessful software projects failed because of inefficient planning of project tasks and human resources [3]. Due to the importance and difficulty of software project planning, there is a growing need for developing effective computer aided tools for software project planning in recent years [4], [5], [6].

To plan a software project, the project manager needs to estimate the project workload and cost and decide the project schedule and resource allocation. For workload and cost estimation, some famous models like COCOMO [7], [8] have been developed and widely used. For scheduling and staffing management, similarly to other projects (e.g., construction projects), management is usually conducted by project management tools and techniques. For example, traditional project management techniques like the program evaluation and review technique (PERT), the critical path method (CPM) [9], and the resource-constrained project scheduling problem (RCPSP) model [10] have been applied in software project planning. Although these methods are important and helpful, they are increasingly considered to be inadequate for modeling the unique characteristics of today's software projects [11], [12], [13], [14]. The main reason is that, differently from other projects, a software project is a people-intensive activity and its related resources are mainly human resources [4]. Different software project tasks require employees with different skills, and skill proficiency of employees significantly influences the efficiency of project execution. As such, assigning employees to the best-fitted tasks is challenging for software project managers, and human resource allocation has become a crucial part in software project planning. Techniques like PERT and CPM lack the consideration of resource allocation and scheduling models like the RCPSP do not consider the allocation of employees with various skills. Therefore, the tools based on these traditional project management techniques usually regard task scheduling and human resource allocation as two separated activities and leave the job of human resource allocation to be done by project managers manually [15], resulting in inefficient resource allocation and poor management performance. Moreover, as the main resources in software development are humans instead of big machines, resources in software projects can usually be allocated in a more flexible way than those in construction or manufacturing projects. In contemporary software projects, it is common that a programmer

• *The authors are with the Department of Computer Science, Key Laboratory of Machine Intelligence and Sensor Network, Ministry of Education, and Key Laboratory of Software Technology, Education Department of Guangdong Province, Sun Yat-Sen University, No. 132, Wai-Huan-Dong Road, Da-Xue-Cheng, Guangzhou 510006, P.R. China. E-mail: chenwn3@mail.sysu.edu.cn, junzhang@ieee.org.*

Manuscript received 15 Dec. 2010; revised 7 Jan. 2012; accepted 8 Feb. 2012; published online 2 Mar. 2012.
Recommended for acceptance by J. Grundy.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-12-0373.
Digital Object Identifier no. 10.1109/TSE.2012.17.

joins multiple module development tasks simultaneously, and it is also possible that he stops his current work and joins the other more critical tasks [3]. If task preemption is properly designed, human resources can be organized in a more efficient way. Some research has pointed out that task preemption can have a significant impact on reducing the time and cost of a software project [15], [16]. However, currently, task preemption is still sparsely considered in software project management models and tools [17]. Most existing models have the assumption that each employee can only be assigned to a single task at one time [18], [19], [20]. This assumption reduces the flexibility of resource allocation in software project planning.

To build more suitable models and tools, traditional project management techniques need to be further extended. One noteworthy approach is to model software project planning as a search-based optimization problem. During the last decade, the idea of formulating software engineering as search-based problems has attracted increasing attention [21], [22]. Various software engineering activities such as cost estimation [23], module clustering [24], design [25], testing [26], [27], and software release planning [28] have been modeled as search-based problems and metaheuristic algorithms have been applied successfully. Search-based approaches may also become a promising way for software project planning.

In the literature, several works have been done on developing search-based approaches for software project planning. Duggan et al. [2] and Barreto et al. [4] built models for the staffing problem of software projects and proposed genetic algorithm (GA) approaches. But, their models only focused on staffing and the problem of task scheduling was not considered. Chang et al. proposed the software project management net (SPMnet) model [12] and the project management net (PM-net) model [11] successively, and then further improved the models to a richer version with a GA [15]. Other GA-based approaches were also proposed in [5] and [29], [30], [31], [32]. In these approaches, a plan is described by a 2D matrix which specifies the workload of each employee on each task. But, as this representation is inadequate for modeling resource conflict, these models all implicitly uses the "Mongolian Horde" strategy [6] that assumes an unlimited number of employees can be assigned to a task and an employee can join an unlimited number of tasks simultaneously, which is usually not the case in practice. Bellenguez and Néron [19], [20] proposed a multiskill scheduling model by extending the traditional RCPSP model. The model considers both the problems of human resource allocation and task scheduling, and takes the skill proficiency of employees and resource conflict into account. Tabu search (TS) [19], branch and bound [20], and GA [18] have been developed for the model. In all of the above-mentioned models, there is an assumption that preemption is not allowed. As discussed before, this assumption reduces the flexibility of human resource allocation for software projects. Task preemption in software projects is only considered in a few studies. In Chang's recent work [6], he improved his previous scheduling model by introducing a 3D matrix representation, specifying the workload assignment of each employee for each task on each time period. Although this representation is much more flexible, it makes the search space very large and suffers from the problem of desultory assignment of workloads. Overall, developing effective approaches for software project scheduling and employee allocation remains a challenging problem and deserves further research.

In this paper, we develop a practical and effective approach for the task scheduling and human resource allocation problem in software project planning with an ant colony optimization (ACO) algorithm. Different from the existing approaches, the proposed method is characterized by the following two features.

First, a representation scheme with a novel event-based scheduler (EBS) is developed. The representation scheme is composed of a task list and a planned employee allocation matrix. The task list defines the priorities of tasks to consume resources, and the planned employee allocation matrix specifies the originally planned workload assignments. In this way, the representation takes both the issues of task scheduling and resource allocation into account. The EBS regards the beginning time of the project, the time when resources are released from any finished task, and the time when employees join or leave the project as events. To generate an actual timetable, the EBS adjusts the workload assignments of employees at events and resource conflict is solved according to the priority defined by the task list. In this way, the proposed scheme is practical and flexible as it enables the modeling of task preemption and resource conflict. At the same time, compared with the 3D matrix representation [6], the proposed scheme reduces the size of the search space and thus accelerates the search process. In addition, as the EBS only makes new assignments at events, it is able to keep the implementation of tasks in a more stable manner.

Second, different from the GA and TS approaches developed in the existing studies, we proposed an ACO approach. ACO was proposed by Dorigo [33], Dorigo and Gambardella [34] in the early 1990s and by now has been successfully applied to various combinatorial optimization problems [35], [36], [37], [38], [39]. As ACO builds solutions in a step-by-step manner and enables the use of problem-based heuristics to guide the search direction of ants, it is possible to design useful heuristics to direct the ants to schedule the critical tasks as early as possible and to assign the project tasks to suitable employees with required skills. Therefore, ACO promises to converge fast and perform well on the considered problem. In the experiments, the proposed method is compared with four other approaches on three real projects and 80 randomly generated instances. Experimental results show that the proposed approach is promising.

The rest of this paper is organized as follows: Section 2 formulates the model for software project task scheduling and human resource allocation. Section 3 reviews the related work of scheduling and staffing for software projects. Section 4 introduces the representation scheme and the EBS. In Section 5, the ACO algorithm is proposed. Section 6 analyzes the experimental results. Conclusions are finally drawn in Section 7.

2 Model Description

This paper considers a software project planning model that addresses both the problems of human resource

allocation and task scheduling. The model is described in detail as follows.

2.1 Description of Employees

Software development is a people-intensive activity. To manage employees, an employee database is needed to record the employees' information of wages, skills, and working constraints. The problem of employee allocation is to assign employees to suitable tasks so that the tasks can be done efficiently. Suppose m employees are involved in the project, for the ith employee ($i = 1, 2, \ldots, m$) the following attributes are considered.

- bs_i—The basic salary for the employee per time period (e.g., month).
- hs_i—The salary for the employee's per-hour normal work.
- ohs_i—The salary for the employee's per-hour overtime work.
- nh—Legal normal working hours per month.
- $maxh_i$—Maximum possible working hours per month of the employee for the project.
- $[join_i, leave_i]$—The time window when the employee is available for the project.
- $\{s_i^1, s_i^2, \ldots, s_i^\Phi\}$—The skill list for the employee, where Φ is the number of skills and $s_i^j \in [0, 5]$ is the proficiency score of the jth skill. Here the skills can be documenting, C++ programming, GUI design, and any other technical abilities. $s_i^j = 0$ means the employee does not have the skill and $s_i^j = 5$ means the employee is masterly on that skill.

Basic salaries are paid to regular employees every month regardless of the workloads the employees devote to the project. On the other hand, per-hour normal work salaries are paid according to the working hours. Usually, there are two types of employees: regular ones and temporary ones. Regular employees have stable basic salaries, while temporary ones do not have basic salaries but have much higher per-hour working salaries.

Let us suppose that the ith employee devotes $hours_i^t$ hours to the project at the tth month ($hours_i^t \leq maxh_i$). If $hours_i^t$ is larger than the legal normal working hours nh, it implies that the employee works overtime for the project. The salary $salary_i^t$ for the ith employee at the tth month is calculated by

$$salary_i^t = \begin{cases} bs_i + hours_i^t \cdot hs_i, & hours_i^t \leq nh, \\ bs_i + nh \cdot hs_i & \\ \quad + (hours_i^t - nh) \cdot ohs_i, & nh < hours_i^t \leq maxh_i, \\ \infty, & hours_i^t > maxh_i. \end{cases} \tag{1}$$

2.2 Description of Tasks

In a software project, tasks can be any activity involved in software construction, for example, class design, programming, and testing. A commonly used technique for task description is the task precedence graph (TPG) [6], [15]. A TPG is an acyclic directed graph $G(T, A)$. The set of nodes $T\{t_1, t_2, \ldots, t_n\}$ corresponds to the set of tasks, where n is the number of tasks in the project. The set of arcs A represents the precedence relations among tasks. An $\mathrm{arc}(i, j) \in A$ means t_i is a direct predecessor task of t_j. An example of a TPG is shown in Fig. 1. Under the precedence constraint defined by the TPG, a task can only start when all of its direct predecessor tasks have finished. The problem of task scheduling is to construct an efficient timetable for the implementation of tasks subject to the precedence constraints.

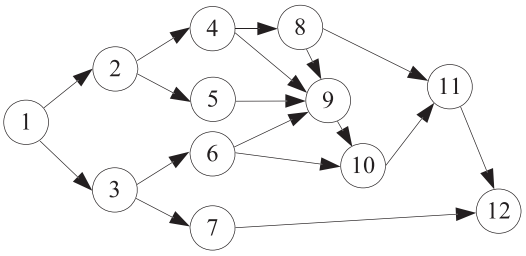

Fig. 1. An example of the TPG.

For a task $t_j (j = 1, 2, \ldots, n)$, the following attributes are considered

- pm_j—The estimated work effort of the task in person months. Several famous methods, for example, the COCOMO models [7], [8], can be adopted for work effort estimation.
- SK_j—The set of skills required by the task.
- $maxhead_j$—The maximum headcount for the task. In application, as too many employees working on the same task will incur higher communication overhead and result in low efficiency, it is necessary to limit the number of employees for a task. According to [6], the maximum headcount can also be estimated based on the COCOMO model.
- $deadline_j$ and $penalty_j$—The deadline and penalty of the task. In practice, it is common to define deadlines for milestone tasks. If the task is delayed, a penalty will be incurred.

Based on the above definitions, suppose wh_{ij}^t is the number of working hours of the ith employee for t_j at the tth month; according to [6], [8], the achievement A_j^t yielded by the employees for t_j at time t can be evaluated by the following steps:

1. The proficiency $prof_{ij}$ of the ith employee for t_j can be evaluated by
$$prof_{ij} = \prod_{id \in SK_j} \frac{s_i^{id}}{5}. \tag{2}$$
2. The total fitness F_j^t of the employees for t_j on the tth month is given by
$$F_j^t = \frac{\sum_{i=1}^m prof_{ij} \cdot wh_{ij}^t}{\sum_{i=1}^m wh_{ij}^t}. \tag{3}$$
3. Convert F_j^t to a cost driver value $V = 8 - round(F_j^t \cdot 7 + 0.5)$, where the value of V belongs to 1-7. $V = 1$ means the employees are the most suitable for the task and vice versa.

4. The achievement A_j^t for t_j on the tth month is calculated by

$$A_j^t = \frac{\sum_{i=1}^{m} wh_{ij}^t}{V}. \quad (4)$$

2.3 Planning Objective

As the software project planning problem involves task scheduling and employee allocation, a plan for a project must specify when the tasks of the project are processed and how the workloads of employees are assigned to the tasks. More specifically, the plan has to determine the start time $start_j$ and the finish time $finish_j$ of each task t_j ($j \in \{1, 2, \ldots, n\}$), and the working hours wh_{ij}^t of all employees $i \in \{1, 2, \ldots, m\}$ to the task t_j during the time window $t \in [start_j, finish_j]$. The plan must satisfy the following constraints:

1. The processing order of tasks must obey the precedence constraint defined by the TPG.
2. The working hours of the ith employee per month must not exceed the limit $maxh_i$, i.e.,

$$\sum_{j=1}^{n} wh_{ij}^t = hours_i^t \leq maxh_i, \quad t = 1, 2, \ldots \quad (5)$$

3. The number of employees assigned to a task t_j is limited by the maximum headcount, i.e.,

$$\sum_{i=1}^{m} sign\left(\sum_{t=start_j}^{finish_j} wh_{ij}^t\right) \leq maxhead_j, \quad \text{where } sign(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases} \quad (6)$$

4. All tasks have to be complete. In other words, for a task t_j, the sum of the achievements for t_j during the time window $[start_j, finish_j]$ must satisfy

$$\sum_{t=start_j}^{finish_j} A_j^t \geq pm_j. \quad (7)$$

This paper considers cost minimization as the objective function, which is given by

$$\min f = \sum_{t=1}^{end} salary_i^t + \sum_{j=1}^{n} penalty_j, \quad (8)$$

where the first item is the salary expenditure and the second item is the total penalty.

3 Related Work Review

This paper intends to propose a representation scheme with a novel event-based scheduler and an ant colony optimization approach for the aforementioned problem. In order to better illustrate why the representation scheme and the EBS are designed, we first briefly review the existing (software) project planning models in this section.

3.1 The Resource Constrained Project Scheduling Problem Model

The RCPSP is a classic model in project management which is NP-hard [10], [40]. It involves scheduling the tasks of a project subject to precedence and resource constraints. To build a schedule, one needs to determine an order of tasks which forms a task list

$$(t_{p_1}, t_{p2}, \ldots, t_{p_n}), \quad (9)$$

where $(p_1, p_2, \ldots, p_n)$ is a permutation of $(1, 2, \ldots, n)$. The task list defines the tasks' priorities to consume resources when resource conflict occurs [10].

Although the RCPSP is important and widely used, it only focuses on the issue of project scheduling and does not take the issue of employee allocation into account. Therefore, the RCPSP is still inadequate for modeling the software project planning problem.

3.2 Employee Allocation Models

Employee allocation problems have attracted a considerable amount of research effort in recent years and various models like nurse rostering [41] and personnel planning [42], [43] have been proposed. There are also some studies that focus on the employee allocation problem in software projects [15], [29], [30], [31], [32]. In these models, the problem of how to assign employees to different workstations (or tasks/time periods) is addressed. The optimization objective is to minimize the number of constraint violations [41], [42], [43] or to minimize project duration and cost [15], [29], [30], [31], [32]. Though the objectives of these models are quite different, they usually describe a plan for the problem by an employee allocation matrix. For example, in [15], the employee allocation matrix is described by

$$\begin{Bmatrix} wh_{11} & wh_{12} & \cdots & wh_{1n} \\ wh_{21} & wh_{22} & \cdots & wh_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ wh_{m1} & wh_{m2} & \cdots & wh_{mn} \end{Bmatrix}, \quad (10)$$

where wh_{ij} means the working hours of the ith employee for t_j.

These employee allocation models are helpful for managing human resources. However, they lack the consideration of task scheduling. To plan a software project, the project manager has to decide not only the allocation of human resources but also the start and finish time of each task. As the working duration of a task is dependent on the employees assigned to the task, separating task scheduling and employee allocation as two independent activities is considered to be unsuitable [6]. In addition, an employee allocation matrix like (10) has no information about how to deal with resource conflict. As a result, the employee allocation models for software project planning [15], [29], [30], [31], [32] have to implicitly assume that a task can be conducted by an unlimited number of employees and an employee can be assigned to an unlimited number of tasks at one time, which is usually not the case in practice [6].

3.3 Multiskill Scheduling Models

Considering the fact that software development involves a group of employees with different skills, Bellenguez and Néron [19], [20] extended the traditional RCPSP model to

propose a multiskill project scheduling model for software project planning. The model follows the framework of the RCPSP for task scheduling and regards different combinations of employees as different alternative modes for the implementation of a task. Tabu search and branch and bound algorithms were also developed for the problem. In a recent work [18], Yannibelli and Amandi further developed a representation scheme for the problem by combining the task list representation (9) and the employee allocation matrix (10). Compared with the models aforementioned, as the multiskill scheduling model is derived from the original RCPSP and couples the modeling of employees with various skills, it provides a more practical way for software project planning as it can deal with both the problems of task scheduling and employee allocation. But, since the model assumes that an employee can only be assigned to a task at one time and no task preemption is allowed, the flexibility of human resource allocation is reduced. Under such restriction, if any one of the employees assigned to a task is busy with other activities, the whole team has to wait until that employee is released. This situation may reduce the resource efficiency of the project.

3.4 The Time-Line-Based Model

To model the task scheduling and human resource allocation problem in a more flexible way, Chang et al. [6] considered task scheduling and employee allocation together and developed a time-line-based model. Different from the previous approaches, the time-line-based model introduces the time-line axis to solution representation and thus a feasible plan is described as a 3D employee allocation matrix:

$$\left\{ \begin{Bmatrix} wh_{11}^1 & \cdots & wh_{1n}^1 \\ \vdots & \ddots & \vdots \\ wh_{m1}^1 & \cdots & wh_{mn}^1 \end{Bmatrix}, \begin{Bmatrix} wh_{11}^2 & \cdots & wh_{1n}^2 \\ \vdots & \ddots & \vdots \\ wh_{m1}^2 & \cdots & wh_{mn}^2 \end{Bmatrix} \cdots, \begin{Bmatrix} wh_{11}^{end} & \cdots & wh_{1n}^{end} \\ \vdots & \ddots & \vdots \\ wh_{m1}^{end} & \cdots & wh_{mn}^{end} \end{Bmatrix} \right\}, \tag{11}$$

where wh_{ij}^t is the working hours of the ith employee for t_j at time t.

The 3D representation is able to overcome the disadvantages of the RCPSP and the employee allocation models. But it also brings in two new problems. First, as workloads are assigned period-by-period instead of task-by-task, the plans produced by this model may assign two completely different groups of employees to the same task in different periods. As a result, the task may be implemented in a desultory manner, which is inefficient. Second, as the representation size of a schedule is enlarged from a 2D matrix to a 3D one, the search space of the problem significantly increases.

4 Representation and the Event-Based Scheduler

To overcome the deficiencies of the above-mentioned models, this paper proposes a representation scheme with a novel event-based scheduler. Similarly to the representation in Yannibelli and Amandi's recent work [18] for the multiskill scheduling problem, we combine the task list representation (9) and the employee allocation matrix representation (10) so that both the problems of task scheduling and human resource allocation are addressed. The representation scheme is given by (12)

$$\begin{bmatrix} \text{Task list:} & (t_{p_1}, t_{p_2}, \ldots, t_{p_n}) \\ \text{Planned employee allocation matrix:} & \begin{Bmatrix} pwh_{11} & pwh_{12} & \cdots & pwh_{1n} \\ pwh_{21} & pwh_{22} & \cdots & pwh_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ pwh_{m1} & pwh_{m2} & \cdots & pwh_{mn} \end{Bmatrix} \end{bmatrix}. \tag{12}$$

Similarly to (9), the task list in (12) also specifies the priorities of tasks in the schedule. However, differently from Yannibelli and Amandi's scheme [18] that directly uses the employee allocation matrix (10), the proposed representation scheme in (12) uses the planned employee allocation matrix. More specifically, the wh_{ij} in (10) is the actual working hours of the ith employee for t_j. But the pwh_{ij} in (12) is only the originally planned working hours of the ith employee for t_j. Because the multiskill scheduling model has various restrictions and reduces the flexibility of human resource allocation, to relax such restrictions and enable task preemption we no longer fix the working hours of the ith employee for t_j during the processing course of t_j as pwh_{ij}. Instead, the actual working hours wh_{ij}^t during the time $t \in [start_j, finish_j]$ should be adjusted according to the urgency of tasks and the amount of available resources. To adjust the planned working hours pwh_{ij} to the actual working hours wh_{ij}^t, we propose the EBS in this paper.

The EBS is characterized by making new assignments at events. We regard the time t as an event if t satisfies any one of the following three conditions: 1) $t = 1$ is the beginning of the project, 2) any employee joins or leaves the project at t, or 3) any task just finished in the previous time period and the corresponding resources become released and available at t. The EBS adjusts a plan in the form of (12) into an actual timetable by two rules. First, if there is resource conflict between two tasks, the task that appears earlier in the task list has a higher priority to use the resource. That is, assuming that the ith employee is originally planned to simultaneously dedicate pwh_{ij} and pwh_{ik} of his working hours to t_j and t_k, respectively, if $pwh_{ij} + pwh_{ik} > maxh_i$, the employee will first dedicate his working hours to the task with a higher priority. Second, new workload assignments are only made when events occur. If no employees join or leave the project or no human resource is released by the tasks just finished, the workload assignments remain the same as the previous time period.

The pseudocode of the EBS is given in Fig. 2. At the beginning, the start time of the project and the time when employees join or leave the project are set as events. Then the scheduler assigns workloads in chronological order with the growing of time t. If t is an event, the scheduler reassigns the actual workloads according to the priority of a task defined by the task list and the originally planned workloads given in (12) (lines 6-16). Otherwise, if t is not an

```
procedure  scheduler      //EBS
01    initialize the number of available human resources during the whole scheduling window;
02    set the beginning time and the time when employees join or leave the project as events;
03    t = 1;
04    while the project is not finished
05        if t is an event
06            find the tasks that can be implemented at time t and arrange these tasks into a sequence seq
                according to the order defined by the task list;
07            while seq is not empty
08                set t_j as the first task in seq and remove t_j from seq;
09                for i=1 to m                    //for every employee
10                    if the planned working hours pwh_ij is not larger than the remaining available working
                        hours of the i-th employee at t
11                        wh_ij^t = pwh_ij;
12                    else
13                        wh_ij^t is set to the remaining available working hours of the i-th employee at t;
14                    end if-else
15                end for
16            end while
17            (local refinement: let regular employees devote all of their normal working hours to the project.)
18        else
19            the workload assignments are the same as those at t-1;
20        end if-else
21        evaluate the completion situation of the tasks at time t;
22        if there are any tasks finished at time t
23            set the time t+1 as an event;
24            (local refinement: release redundant working hours of employees.)
25        end if
26        t = t+1;
27    end while
end procedure
```

Fig. 2. Pseudocode of the EBS.

event, the workloads at t remain the same as those at $t-1$ (line 19). After allocating the actual workloads at t, the achievement A_j^t of t_j at t is calculated by (2)-(4) (line 21). If there is any task finished at t, the time $t+1$ is set as an event because the resources assigned to these tasks can all be released at $t+1$. These steps run repeatedly until all tasks of the project have been finished.

In Fig. 2, there are local refinement steps in lines 17 and 24. The local refinement steps are designed to further improve performance of the plan. It is based on two ideas.

First, in practice, regular employees are usually expected to work more for the project so that the costs for hiring extra employees can be saved. Therefore, the refinement step in line 17 is to let regular employees dedicate their normal working hours to the project. More specifically, for all regular employees i, if the assignment in lines 7-16 satisfies $hours_i^t = \sum_{j=1}^{n} wh_{ij}^t > 0$, $hours_i^t < maxh_i$, and $hours_i^t < nh$, it means that the ith employee is planned to work for the project at time t but not all of his normal working hours are dedicated. In this case, according to the order defined by the task list, we find the first task t_j from the task list that satisfies $wh_{ij}^t > 0$ and update wh_{ij}^t to

$$wh_{ij}^t = \begin{cases} wh_{ij}^t + (maxh_i - hours_i^t), & \text{if } maxh_i < nh, \\ wh_{ij}^t + (nh - hours_i^t), & \text{if } maxh_i \geq nh. \end{cases} \quad (13)$$

In this way, the ith employee can dedicate all of his normal working hours nh to the project.

Second, if a task t_j is finished at time t, it is possible that there are redundant assignments at t. In other words, if some workloads are released, task t_j may still be finished at t. To reduce cost, it is necessary to reduce such redundant assignments in line 24 in Fig. 2. Redundant assignments are released as follows.

Step a: Arrange the employees that work for t_j at time t (i.e., $wh_{ij}^t > 0$) in ascending order of their proficiency scores $prof_{ij}$ in task t_j. Proficiency scores are defined in (2).

Step b: Select the first employee from the order given in *step a* and denote the employee as u.

Step c: Repeatedly set $wh_{uj}^t = wh_{uj}^t - nh \cdot unitpercent\%$ and test whether t_j can be complete at time t under this assignment until $wh_{uj}^t = 0$ or t_j cannot be complete. Here, although the working hours of employees can be continuous variables, setting the working hours to a very precise value like 159.5 hours/month is meaningless. Therefore, we only test the situation when a certain percentage *unitpercent* of the normal working hours are reduced. According to the discussion of workload discretization in [6], we set $unitpercent = 25$ in this paper.

Step d: If $wh_{uj}^t = 0$, set the next employee in the order given in *step a* as the employee u and go to *step c*. If t_j cannot be complete at time t, reset $wh_{uj}^t = wh_{uj}^t + nh \cdot unitpercent\%$ to make sure that t_j can be finished, and the refinement procedure is ended.

Compared with the RCPSP and the employee allocation models, as the proposed representation scheme combines the task list and the employee allocation matrix it is able to deal with both the issues of task scheduling and employee allocation. In addition, with the EBS, the problems of task preemption and resource conflict can be addressed. Compared with the time-line-based model [6] that uses a 3D matrix with size $m \times n \times end$, the proposed scheme only

uses a task list of length n and a matrix of size $m \times n$ for representation. In other words, the proposed scheme reduces the representation size from $O(m \cdot n \cdot end)$ to $O(m \cdot n)$, and thus reduces the size of search space. As new assignments are only made at events, the project can be executed in a more stable way.

More fundamentally, an essential difference between the proposed scheme and the existing methods is the granularity of planning. The RCPSP and the employee allocation models make plans with the granularity to task. Thus, they can only make plans with each task having a fixed workload assignment and cannot deal with task preemption. On the other hand, the time-line-based model makes plans with the granularity to each time period. Though the model is flexible, the plan is too fine-grained and thus the problems of a large search space and desultory workload assignments are incurred. In contrast, the proposed scheme makes plans with the granularity to event. In this way, the scheme has sufficient information for modeling task preemption and reduces the search space as well.

5 Ant Colony Optimization Approach

5.1 Overview of the ACO Algorithm

To solve the software project planning problem, this paper proposes an ACO approach. The underlying idea of ACO is to simulate the foraging behavior of ants. When ants search for food, they usually deposit a special chemical on the path they travel through. This kind of chemical, which is called pheromone, serves as a medium for ants to communicate with each other. By sensing the concentration of pheromone, other ants can follow the path to find the food. Inspired by this swarm intelligence phenomenon, ACO was developed by Dorigo et al. [33], [34] and has been successfully applied to various optimization problems. An ACO algorithm works by dispatching a group of artificial ants to build solutions to the problem iteratively. In general, an ACO algorithm can be viewed as the interplay and the repeated execution of the following three main procedures [39]:

1. *Solution construction*—During each iteration of the algorithm, a group of ants set out to build solutions to the problem. Each ant builds a solution in a constructive manner by selecting components step by step to form a complete solution. The selections are made according to pheromone and heuristic information. In ACO, pheromone is a record of the past search experience of ants for guiding the following ants to make decisions. The components belonging to the best solutions found by the previous ants usually accumulate more pheromone, attracting more ants to select in future iterations. Heuristic is some problem-dependent information that helps ants to have higher probabilities to select promising components in the solution construction procedure.
2. *Pheromone management*—Along with the solution construction procedure, pheromone values are updated according to the performance of the solutions built by ants. Ants tend to deposit more pheromone to the components of better-performed solutions.
3. *Daemon actions*—Daemon actions mean the centralized operations that cannot be done by single ants. In the design of ACO algorithms, daemon actions are optional, but many existing ACO variants use different kinds of daemon actions to improve performance [39]. One commonly seen daemon action is the local search procedure.

By now, various ACO variants have been developed [39]. Two of the best performing ACO variants include ant colony system (ACS) [34] and max-min ant system (MMAS) [44]. In this paper, we follow the ACS variant to develop the ACO approach to the considered software project planning problem. The main characteristics of the ACS are in two aspects. First, in the solution construction procedure, the ACS applies a pseudorandom proportional selection rule which aggressively biases selecting the components with the maximum pheromone and heuristic values. In this way, the ACS strongly exploits the past search experience of ants and has a fast convergence speed. Second, in the pheromone management procedure, ACS has two pheromone updating rules, namely, the global updating and the local updating. The global updating rule makes the components corresponding to the best-so-far solution become more attractive. The local updating rule reduces the pheromone on the components just selected by ants to increase the search diversity of the algorithm.

With the above procedures and using the ACS variant, the overall flowchart of the proposed ACO approach is shown in Fig. 3. At the beginning, parameters of the algorithm are initialized. In each iteration, ants set out to build plans for the problem. Since a plan for the considered problem includes the task list and the planned employee allocation matrix, the solution construction procedure in the proposed algorithm has two steps: 1) construction of the task list and 2) construction of the employee allocation matrix. Pheromone values are updated by the global and the local updating rules. Finally, a local mutation procedure is designed as the local search to further improve the performance of the algorithm.

5.2 Solution Construction

5.2.1 Construction of Task List

A task list is an order of tasks $(t_{p_1}, t_{p_2}, \ldots, t_{p_n})$ that satisfies the precedence constraints defined by the TPG. Here we first define the pheromone and the heuristic for task list construction. Then the procedure for an ant to build a task list will be described in detail.

Pheromone. To build a task list, an ant has to determine an order of the tasks. In existing studies of ACO for scheduling problems like shop scheduling problems and the RCPSP, there are mainly two types of pheromone models for finding an order of tasks. The first type is the absolute position model that defines the pheromone of putting a task t_j to the kth position of the task list as $\tau_t(j,k)$ [45]. Since the absolute position model can only indicate the desirability of putting a task to a certain position, it is found that the model can lead to highly suboptimal solutions [39]. To improve this model, Merkle et al. [35] developed a summation rule. The summation rule uses the sum of the pheromone values of putting a task t_j to the positions 1 to k instead of simply using $\tau_t(j,k)$ in the selection. In this way, the deficiency of the absolute position model is overcome

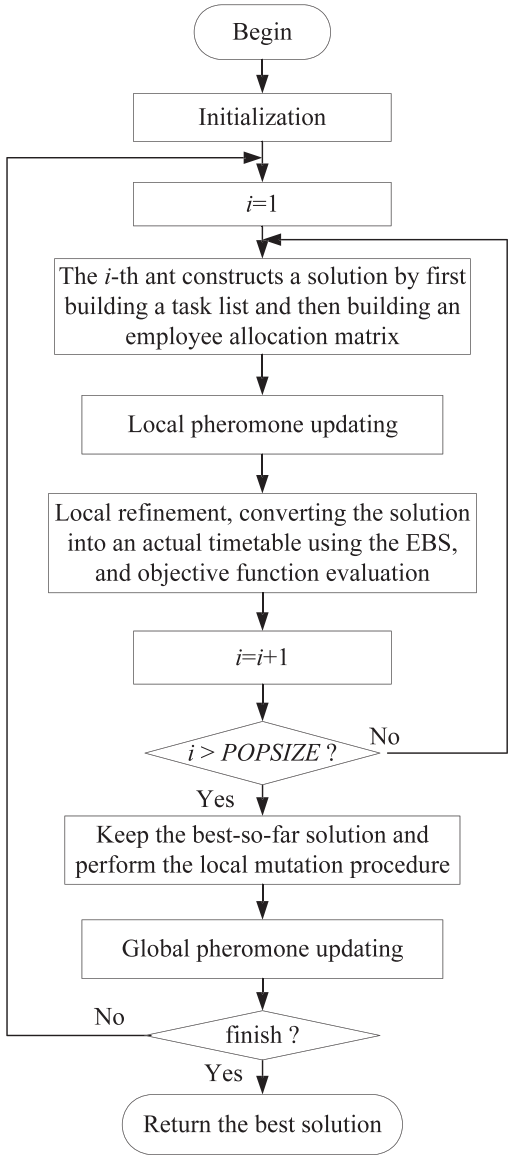

Fig. 3. Flowchart of the ACO algorithm.

and the resulting approach is effective. The second type of pheromone models is the relative position model that defines the pheromone on the link between two tasks and uses the pheromone to indicate the desirability of putting one task directly after the other one [37], [46]. The disadvantage of this model is that the relative positions of two tasks may become meaningless when the two tasks are reconcilable with respect to resource (machine) consumption. To improve this model, Blum and Sampels [47], Blum [48] developed a relation-learning model that only defines pheromone on the links between two related tasks in the same group or processed by the same machine for shop scheduling problems. In this way, the relation-learning model overcomes the deficiency of the conventional relative position model and achieves promising performance.

For the considered software project planning problem, since one task can be assigned to several employees, one employee can undertake several tasks simultaneously, and skill proficiency is considered, it is more difficult to define related tasks for the relation-learning model. Therefore, we adopt the absolute position model with the summation rule in the proposed approach. In other words, the pheromone $\tau_t(j,k)$ in the proposed approach means the desirability of putting the task t_j to the kth position. The summation rule will be presented in detail in the construction procedure of the task list.

Heuristic. The minimum slack (MINSLK) heuristic [40], [49] is adopted for task list construction. A task with a relatively smaller MINSLK implies that this task is more urgent. The MINSLK for a task t_j, which is denoted as $MINSLK_j$, can be estimated as follows:

Step a: Estimate the shortest possible makespan of each task.

To estimate the shortest possible makespan of t_j, we select $maxhead_j$ most proficient employees of t_j and assume these $maxhead_j$ employees dedicate all of their working hours to t_j. Here $maxhead_j$ is the maximum headcount for t_j and the proficiency of employees for t_j can be evaluated by (2). With this assumption, the shortest possible makespan of t_j can be estimated using (3) and (4).

Step b: Based on the shortest possible makespan of each task, the earliest start time and the latest start time of each task can be evaluated, and the MINSLK is calculated by the difference between the latest start time and the earliest start time of the task [49]. Then the heuristic for task t_j (denoted as $\eta_t(j)$) is given by

$$\eta_t(j) = 1/MINSLK_j. \tag{14}$$

Construction procedure. To build a feasible task list, each ant maintains an *eligibleSet* of the tasks that satisfy the precedence constraint. The construction includes the following steps:

Step a: Put the tasks that can be implemented at the beginning of the project (i.e., the tasks that do not have any precedence tasks) into the *eligibleSet*.

Step b: For $k = 1$ to n, process the following substeps *b-1* and *b-2*, repeatedly:

Step b-1: Select a task from *eligibleSet* and put the task to the kth position of the task list. In the selection rule, at first a random number q uniformly distributed in $[0,1]$ is generated and is compared with a parameter q_t. If $q < q_t$, then the task t_j from the *eligibleSet* with the largest value of $\sum_{l=1}^{k} \tau_t(j,l) \cdot \eta_t(j)$ is chosen to put to the kth position. Otherwise, the task is selected using the roulette wheel selection scheme. The probability $\Pr(j,k)$ of selecting the task t_j to the kth position in the roulette wheel selection scheme is given by

$$\Pr(j,k) = \begin{cases} \dfrac{\sum_{l=1}^{k} \tau_t(j,l)\cdot\eta_t(j)}{\sum_{t_u \in eligibleSet} \sum_{l=1}^{k} \tau_t(u,l)\cdot\eta_t(u)}, & \text{if } t_j \in eligibleSet, \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

This selection rule is designed based on the pseudorandom proportional rule in the ACS [34] and the summation rule [35]. The characteristic of the pseudorandom proportional rule is that ants have a probability q_t of selecting the task with the maximum pheromone and heuristic values directly so that the algorithm can strongly exploit the past search experience of ants. The characteristic of the summation rule is to use the sum of the pheromone values of putting t_j to

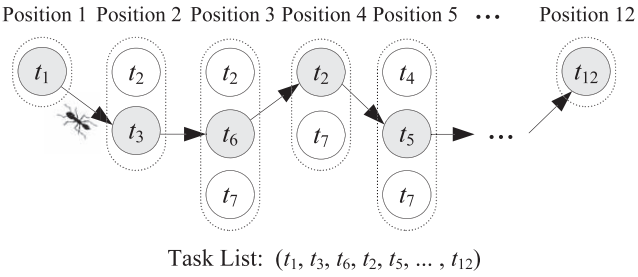

Fig. 4. An example of task list construction based on the TPG given in Fig. 1.

positions 1 to k instead of simply using the pheromone value $\tau_t(j,k)$ in the rule. In this way, the deficiency of the absolute position pheromone model can be overcome.

Step b-2: Update the *eligibleSet* by removing the selected task from *eligibleSet* and adding new feasible tasks that satisfy the precedence constraint into *eligibleSet*.

After *Steps b-1* and *b-2* repeat n times, a feasible task list is built. An example of how an ant builds a task list (based on the TPG given in Fig. 1) is illustrated in Fig. 4.

5.2.2 Construction of Employee Allocation Matrix

The employee allocation matrix in (12) specifies the originally planned working hours of employees to tasks. In general, for the working hours of the ith employee for the task t_j, the domain of pwh_{ij} is $[0, maxh_i]$. However, as setting the workload assignment to a very accurate value is not meaningful in practice, according to Chang et al. [6], [15] it is possible to discretize the search domain of pwh_{ij}. Here we use $nh \cdot unitpercent\%$ as the unit of working hours for discretization. If $unitpercent = 25$, the search domain of pwh_{ij} becomes $\{0, 25\%\ nh, 50\%\ nh, \ldots, maxh_i\}$. (Note that if $maxh_i > nh$, the percentage can be larger than 100 percent.) With the maximum headcount constraint, at most $maxhead_j$ employees can be selected to devote their working hours to t_j. The working hours of other unselected employees is set to 0.

Pheromone. Here two kinds of pheromone are adopted for employee allocation matrix construction. The pheromone of choosing the ith employee to work for the task t_j is denoted as $\tau_{e1}(i,j)$. The pheromone of assigning k ($k = 25\%\ nh,\ 50\%\ nh, \ldots, maxh_i$) of the ith employee's working hours to t_j is denoted as $\tau_{e2}(i,j,k)$.

Heuristic. The heuristic of choosing the ith employee to work for the task t_j is denoted as $\eta_e(i,j)$. We define $\eta_e(i,j)$ as

$$\eta_e(i,j) = prof_{ij}/hs_i. \tag{16}$$

The meaning of this heuristic definition is the rate between the proficiency $prof_{ij}$ of the ith employee for the task t_j and the hour salary of the employee. An employee with a lower salary and a higher proficiency score for t_j is more probable to be chosen to work for t_j.

Construction procedure. The construction of the employee allocation matrix is based on the following steps:

Step a: Set all values in the employee allocation matrix to 0.

Step b: For each task t_j $(j = 1, 2, \ldots, n)$, assign the workloads for t_j by the following substeps:

Step b-1: Evaluate the value of $\tau_{e1}(i,j) \cdot \eta_e(i,j)^\beta$ for all employees, where β is a parameter.

Step b-2: Select an employee u from the set of employees which have not yet been chosen for t_j using the pseudorandom proportional rule in the ACS [34]. A random number q uniformly distributed in $[0,1]$ is first generated and is compared with a parameter q_{e1}. If $q < q_{e1}$, then the employee u from the eligible set *Eligible* of employees with the largest value of $\tau_{e1}(u,j) \cdot \eta_e(u,j)^\beta$ is selected for t_j. Otherwise, the employee is selected using the roulette wheel selection scheme. The probability $\Pr(u,j)$ of selecting the employee u for t_j is given by

$$\Pr(u,j) = \begin{cases} \frac{\tau_{e1}(u,j)\cdot\eta_e(u,j)^\beta}{\sum_{i\in Eligible}\tau_{e1}(i,j)\cdot\eta_e(i,j)^\beta}, & \text{if } u \in Eligible, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

If employee u is selected, then this employee will be removed from the *Eligible* set.

Step b-3: Assign the working hours for the employee selected to work for t_j using the pseudorandom proportional rule. The selection is only based on the pheromone values τ_{e2}. A random number q uniformly distributed in $[0,1]$ is first generated and is compared with a parameter q_{e2}. If $q < q_{e2}$, then we set $pwh_{ij} = k$, where $k \in \{25\%nh, 50\%nh, \ldots, maxh_i\}$ and k has the highest pheromone value $\tau_{e2}(i,j,k)$. Otherwise, the selection is made using the roulette wheel selection scheme. The probability $\Pr(i,j,k)$ of setting $pwh_{ij} = k$ is given by

$$\Pr(i,j,k) = \frac{\tau_{e2}(i,j,k)}{\sum_{v\in\{25\%nh,50\%nh,\ldots,maxh_i\}}\tau_{e2}(i,j,v)}, \quad k \in \{25\%nh, 50\%nh, \ldots, maxh_i\}. \tag{18}$$

Step b-4: If the number of the employees selected to work for t_j reaches $maxhead_j$, the workload assignment process for t_j is finished. Otherwise, go to *Step b-2* to select other employees to work for t_j.

After the workloads for all tasks have been assigned, the procedure for building the employee allocation matrix is complete.

5.3 Pheromone Management

At the beginning, all pheromone values are set to an initial value $\tau_{initial}$, where

$$\tau_{initial} = \frac{1.0}{\sum_{i=1}^{m}[bs_i + maxsalary_i]\cdot(deadline + n)}. \tag{19}$$

Here $maxsalary_i$ is the maximum possible salary of the ith employee per time period,which can be calculated by

$$maxsalary_i = \begin{cases} bs_i + maxh_i \cdot hs_i, & 0 \le maxh_i \le nh, \\ bs_i + nh \cdot hs_i + (maxh_i - nh)\cdot ohs_i, & maxh_i > nh. \end{cases} \tag{20}$$

The denominator in (19) is actually an upper bound estimation of the objective function value. The upper bound is estimated by assuming all employees contribute all of their working hours to the project during every time period and the project is delayed by n time periods (months) with respect to a predefined *deadline*. Here, n is a rough worst-case estimation that assumes each task leads to a one-month delay for the project.

During the construction of the task list and employee allocation matrix, immediately after a selection is made the local pheromone updating rule is applied to reduce the corresponding pheromone values so that the following ants can have better chances of selecting other possible choices. In this way, the algorithm manages to maintain diversity. Suppose the task t_j is selected to the kth position of the task list; the corresponding pheromone is updated by

$$\tau_t(j,k) = (1-\rho)\cdot\tau_t(j,k) + \rho\cdot\tau_{initial}, \tag{21}$$

where ρ is a parameter. Similarly, if the ith employee is assigned to the jth task with k working hours, the corresponding pheromone is updated by

$$\tau_{e1}(i,j) = (1-\rho)\cdot\tau_{e1}(i,j) + \rho\cdot\tau_{initial}, \tag{22}$$

$$\tau_{e2}(i,j,k) = (1-\rho)\cdot\tau_{e2}(i,j,k) + \rho\cdot\tau_{initial}. \tag{23}$$

At the end of each iteration, additional pheromone is deposited to the components associated with the best-so-far plans found by the algorithm. All the pheromone values associated with the best-so-far plan are updated by

$$\tau_t(j,k) = (1-\rho)\cdot\tau_t(j,k) + \rho\cdot\Delta\tau, \tag{24}$$

$$\tau_{e1}(i,j) = (1-\rho)\cdot\tau_{e1}(i,j) + \rho\cdot\Delta\tau, \tag{25}$$

$$\tau_{e2}(i,j,k) = (1-\rho)\cdot\tau_{e2}(i,j,k) + \rho\cdot\Delta\tau, \tag{26}$$

where $\Delta\tau = 1/f$ is the reciprocal of the objective function value of the plan. Based on these definitions, as the denominator in (19) is an upper-bound estimation, $\tau_{initial}$ is the lower-bound of all pheromone values. With the evolution of the algorithm, the cost of the best-so-far plan found by the algorithm becomes lower and lower. Thus, the value of $\Delta\tau$ becomes larger. In other words, more pheromone will be added to the components of the best-so-far plan to make them more attractive for the ants in the following iterations.

5.4 Local Mutation Procedure

The considered problem is intractable as both the task list and the employee allocation matrix have to be optimally decided. In particular, although there are effective heuristics for guiding the construction of task list and the selection of employees, it is difficult to find an effective heuristic that can tell us how many working hours an employee works for a specific task is the most suitable. Therefore, to further improve the performance of the ACO approach, it is helpful to couple it with a local search procedure.

Inspired by the mutation operator used by the GA for the multiskill problem [18], a simple local mutation procedure is designed. The procedure is performed by mutating the best-so-far solution found by ACO using either of the following mutation operators.

Operator 1—*Task list mutation*: First, a task t_j is randomly selected from the task list. Then a destination position is randomly generated. The mutation is performed by moving the selected task t_j toward the destination position in the task list as close as possible without violating the precedence constraints.

Operator 2—*Employee allocation matrix mutation*: First a task t_j is randomly selected. Then an employee i who has been assigned to work for this task is randomly selected and his working hours for this task are reset to $pwh_{ij} = 0$. Finally, a new employee u who has not been selected to work for the task is randomly selected and his working hours for the task pwh_{uj} are randomly set to a value from $\{25\%nh, 50\%nh, \ldots, maxh_u\}$.

During each iteration, *mutatenum* of mutated solutions are generated based on either of the above mutation operators. If any mutated solution is better than the current best-so-far solution, then the best-so-far solution will be replaced.

6 Experimental Studies

6.1 Experimental Settings

In the experiments, we test the proposed method on three real instances and 80 randomly generated instances. The three real instances are derived from business software construction projects for a department store company. The TPG in Fig. 1 belongs to one of the real projects. In order to guarantee that the randomly generated instances correctly capture the characteristics of real-world projects, we generate the instances using the following strategies:

1. The project networks of the instances are derived from the PSPLIB [50]. The benchmark instances in PSPLIB have been widely used in project management studies in various fields (including software project management) [18], [51]. Thus, we consider these benchmark project networks to be useful for capturing the task precedence relations of software projects.
2. Based on our empirical observation, we consider four types of employees in a project team: regular elite staff, regular normal staff, temporary experts, and temporary normal staff. Elite staff are experts on one or more skills and their salaries are high. Normal staff are good at one or two skills and they are in the majority of the team. Temporary staff are hired temporarily for the project and we only need to pay for what they contribute to the project. Under this classification, in our random instance generator, the properties of an employee are generated randomly within a predefined domain according to his staff type. After the properties of all staff are generated, a feasibility check is run to ensure the skills required by the project have been covered by the employees.
3. The workload (in person months) of each task of the project is given randomly and the maximum headcount for the task is evaluated based on the COCOMO model [6].

Based on these strategies, we generate 80 random instances by using 80 different project networks from the PSPLIB. The basic information of the project networks is given in Table 1. We name the instances after the sources of their project networks from the PSPLIB. In other words, the instance "c159_1" in our experiment means that the project network of this instance comes from the benchmark instance "c159_1" in the PSPLIB.

TABLE 1
Information of the Test Instances

name of instance	task num	skill num	employee num	max num of solutions
real1	12	5	10	50000
real2	15	5	10	50000
real3	15	5	10	50000
c159_1-- c159_10	18	5	10	50000
j301_1 -- j301_10	32	5	10	100000
j302_1 -- j302_10	32	5	15	100000
j303_1 -- j303_10	32	10	20	100000
j601_1 -- j601_10	62	5	10	200000
j602_1 -- j602_10	62	5	15	200000
j603_1 -- j603_10	62	10	20	200000
j901_1 – j901_10	92	8	15	300000

The parameters of the proposed ACO are set as follows: The number of ants is $POPSIZE = 10$, the pheromone updating rate ρ in (21)-(26) is set to 0.1, and the parameter β in the employee allocation matrix generation procedure is set to 2. These parameters can significantly influence the performance of ACO. We have tested different configurations for these parameters and found that the original configurations suggested by Dorigo and Gambardella [34], Dorigo and Stützle [39] still contribute to good performance for the proposed ACO approach. There is also a local mutation procedure in the proposed ACO algorithm and the number of mutated solutions generated in each iteration $mutatenum$ is a parameter. In the experiment, we find that if $mutatenum < 5$, the local mutation procedure can hardly cause effects. On the other hand, if $mutatenum > 30$, the local mutation procedure consumes too much computational effort so that the convergence speed of the algorithm is influenced. Overall, we find that $mutatenum \in [5, 30]$ is suitable for the problem and we empirically set $mutatenum = 10$ in the experiment. In addition, there are also three parameters, q_t, q_{e1}, and q_{e2}, in the pseudorandom proportional rule developed in the ACS [34] in the solution construction procedure. These parameters are used to control the degree of aggressiveness in the selection rules of the algorithm. If we set q_t, q_{e1}, and $q_{e2} = 0$, the selection rules become the conventional roulette wheel selection scheme. If we set q_t, q_{e1}, and q_{e2} to larger values, the algorithm strongly exploits the past search experience and exhibits a fast-converging manner. But the search diversity of the algorithm is decreased. To find a good tradeoff between convergence and diversity, we study the configurations for q_t, q_{e1}, and q_{e2} empirically and the results are shown in Fig. 5. For the parameter q_t used in the construction of task list and the parameter q_{e2} used in the assignment of working hours, we find that setting q_t, $q_{e2} = 0.5$ can generally yield the best performance. For the parameter q_{e1} used in the selection of employees, we find that the aggressive choice $q_{e1} = 0.9$ is helpful for the algorithm to assign each activity to a suitable group of employees quickly, and thus results in good performance. Thereby, we use the configuration q_t, $q_{e2} = 0.5$ and $q_{e1} = 0.9$ in the proposed ACO approach.

This paper compares the proposed ACO algorithm with the tabu search (TS) algorithm for the multiskill scheduling problem [19], the knowledge-based genetic algorithm (KGA) proposed recently for multiskill scheduling [18], and the time-line-based GA that uses a 3D representation (we denoted as 3dGA in the following text) [6]. The reason for selecting these algorithms for comparison is that all these algorithms can deal with both the problems of task scheduling and human resource allocation in software project planning. In these three algorithms, the TS and the KGA do not allow task preemption, and only the 3dGA allows it. But as the KGA uses a similar representation scheme to the proposed ACO algorithm that combines a task list and an employee allocation matrix, we further develop a KGA-p algorithm by coupling the KGA with the event-based scheduler proposed in this paper. With the EBS, KGA-p can also deal with task preemption. We also compared the KGA-p algorithm with the proposed ACO, and the comparison results can tell whether the EBS is effective and whether the ACO performs well.

The parameters of the algorithms in the comparison are set according to the corresponding references. The TS uses a greedy algorithm to generate a promising initial solution for the search. The parameters of the KGA and KGA-p are set as follows: Population size is 50, mutation probability is 0.05, and crossover probability is 0.8. The parameters of 3dGA are set as follows: Population size is 200, crossover probability is 0.7, and mutation probability is 0.001.

In the experiment, all algorithms are run 30 independent times for each instance. During each run, the algorithm stops when the number of plans built by the algorithm reaches the predefined maximum number of solutions given in Table 1.

6.2 Comparison of the Costs

The objective function considered in this paper is to minimize the cost of the software project. Therefore, we first compare the performance of the proposed method with other approaches in terms of cost. For all test instances, the best and mean results (averaged over 30 runs) found by the algorithms are tabulated in Table 2. To make the comparison

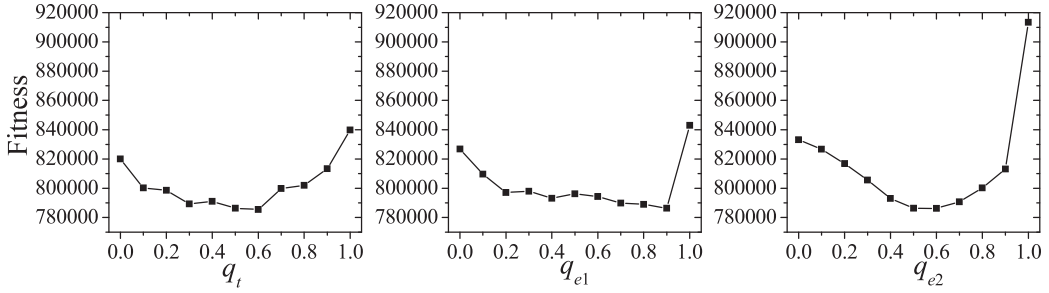

Fig. 5. Analysis of the configurations for the parameters q_t, q_{e1} and q_{e2} on the instance c159_1.

TABLE 2
Comparison of the Results on the 83 Instances

	best	mean	t-test	best	mean	t-test	best	mean	t-test	best	mean	t-test	best	mean	t-test
instance	Real 1			Real 2			Real 3								
ACO	259809	266197		512200	530533		577025	630139							
ACO-L	**252375**	**260178**		**496550**	**517998**		**542675**	**606798**							
3dGA	266115	282722	**-11.34#**	591350	640146	**-26.74#**	815100	919364	**-35.44#**						
KGA	283522	310093	**-22.50#**	621250	670898	**-31.25#**	711050	768084	**-16.81#**						
KGA-p	261242	274812	**-2.81#**	511200	541839	-1.39	595325	652988	**-2.11#**						
TS	294565	308124	**-17.33#**	573375	596403	**-11.84#**	621625	666213	**-3.70#**						
instance	c159_1			c159_2			c159_3			c159_4			c159_5		
ACO	753400	786265		1360400	1401978		1003966	1094766		1110700	1169491		543000	575610	
ACO-L	**710975**	**756210**		**1246025**	**1305303**		**965875**	**1064013**		**1069150**	**1099260**		**490825**	**532246**	
3dGA	874175	962717	**-17.02#**	1621650	1688468	**-25.08#**	1183957	1312157	**-26.32#**	1377925	1440763	**-30.25#**	608525	670890	**-13.27#**
KGA	1111375	1178844	**-42.53#**	1664425	1723322	**-31.05#**	1313557	1365855	**-28.67#**	1432079	1514525	**-38.19#**	796975	857222	**-51.43#**
KGA-p	799500	873272	**-8.79#**	1316925	1380361	2.62*	1014250	1135820	**-7.75#**	1120125	1186222	**-2.14#**	589500	671646	**-11.17#**
TS	955450	1022308	**-27.48#**	1525100	1571659	**-13.61#**	1222825	1263476	**-17.50#**	1272050	1345640	**-17.32#**	734700	759581	**-30.08#**
instance	c159_6			c159_7			c159_8			c159_9			c159_10		
ACO	1036378	1090762		1220250	1319860		1095300	1133316		903625	943255		811150	836719	
ACO-L	**965240**	**1013547**		**1187800**	**1248363**		**1006175**	**1048960**		**824575**	**889999**		**745150**	**778625**	
3dGA	1483800	1526459	**-40.31#**	1493500	1572544	**-33.10#**	1315625	1397360	**-33.14#**	1099125	1208422	**-25.18#**	938600	1016338	**-18.49#**
KGA	1339122	1400875	**-24.97#**	1633350	1729377	**-48.12#**	1347575	1431724	**-45.29#**	1159050	1227170	**-29.19#**	1004600	1069454	**-23.41#**
KGA-p	1073829	1149588	**-5.49#**	1232825	1316986	0.15	1082000	1146241	-1.97	891050	975305	-1.26	788950	837175	-0.08
TS	1225270	1284654	**-11.55#**	1486950	1571364	**-29.61#**	1225900	1290757	**-20.39#**	1058325	1106794	**-15.44#**	895900	941273	**-9.46#**
instance	j301_1			j301_2			j301_3			j301_4			j301_5		
ACO	2268709	2376291		2195575	2258512		2015200	2069197		1647750	1877420		1623841	1688152	
ACO-L	**2157641**	**2245671**		**1973825**	**2054426**		**1647923**	**1762529**		**1330891**	**1550997**		**1432575**	**1544548**	
3dGA	2974387	3077320	**-43.21#**	2586300	2643834	**-31.09#**	2032519	2153152	**-5.71#**	1702391	1970782	**-3.11#**	1515212	1762828	-1.32
KGA	2931191	3088886	**-39.56#**	2640175	2745538	**-35.80#**	2160575	2284343	**-9.61#**	2449342	2541716	**-29.97#**	1997950	2140273	**-33.02#**
KGA-p	2224657	2381628	-0.27	2193325	2291877	**-2.77#**	1807521	1916842	9.52*	1636457	1927979	-1.31	1549396	1701947	-0.88
TS	2710550	2777988	**-27.04#**	2350900	2450318	**-17.61#**	1998100	2047732	2.22*	2264025	2324949	**-16.44#**	1840850	1921330	**-17.39#**
instance	j301_6			j301_7			j301_8			j301_9			j301_10		
ACO	1548375	1604482		1480300	1530845		2356775	2460159		1221675	1257638		1995875	2144448	
ACO-L	**1348675**	**1466237**		**1322575**	**1399064**		**1918100**	**2096573**		**1120450**	**1159175**		**1748950**	**1849638**	
3dGA	1875475	2037463	**-21.11#**	1586200	1727152	**-11.18#**	2553950	2700853	**-8.17#**	1354900	1458149	**-18.11#**	2444900	2572730	**-18.64#**
KGA	2127650	2227533	**-32.47#**	1967025	2144651	**-37.30#**	3016350	3138971	**-40.37#**	1705850	1773004	**-40.04#**	2651075	2788576	**-30.29#**
KGA-p	1616100	1797153	**-9.28#**	1404900	1555055	-1.96	2218600	2434938	2.03*	1205725	1351859	**-9.32#**	1985325	2120685	1.78
TS	1838925	1889046	**-17.61#**	1791600	1860720	**-27.81#**	2779950	2853660	**-21.38#**	1462975	1522710	**-26.85**	2371075	2446461	**-13.04#**
instance	j302_1			j302_2			j302_3			j302_4			j302_5		
ACO	2055675	2117439		1745025	1803068		1598050	1655751		1854725	1901216		1911975	1979661	
ACO-L	**1816625**	**1921462**		**1595825**	**1661564**		**1476325**	**1549330**		**1651150**	**1712303**		**1689625**	**1788699**	
3dGA	2432653	2580190	**-18.47#**	2121575	2231408	**-33.27#**	2142400	2256244	**-41.04#**	2138925	2271104	**-26.41#**	2412026	2518821	**-37.14#**
KGA	2580975	2756045	**-31.32#**	2683350	2830113	**-45.93#**	2276950	2347342	**-50.81#**	2335375	2456444	**-37.58#**	2543075	2674747	**-41.44#**
KGA-p	2027675	2187149	**-4.61#**	1726325	1940658	**-7.40#**	1739275	1839674	**-15.70#**	1819200	1930116	**-2.56#**	1927400	2005479	**-2.48#**
TS	2347100	2432994	**-12.83#**	2251425	2334970	**-35.04#**	1939650	2005595	**-26.41#**	2024225	2097323	**-9.71#**	2229550	2308895	**-20.38#**
instance	j302_6			j302_7			j302_8			j302_9			j302_10		
ACO	1564425	1631275		1580625	1635566		2171075	2289958		1349400	1398531		1800025	1857073	
ACO-L	**1426792**	**1502753**		**1459950**	**1553746**		**1975569**	**2007144**		**1268325**	**1311913**		**1649825**	**1750088**	
3dGA	1809575	2046224	**-28.41#**	1979125	2108344	**-33.22#**	3069444	3204886	**-48.19#**	1771400	1931500	**-25.98#**	1950325	2108248	**-18.83#**
KGA	2112350	2285520	**-38.61#**	2131000	2243315	**-40.95#**	2582231	2732793	**-24.92#**	2127000	2238407	**-40.28#**	2434950	2581148	**-41.32#**
KGA-p	1619325	1764365	**-9.74#**	1719425	1830047	**-14.83#**	2072888	2203188	4.18*	1446275	1614811	**-17.21#**	1810442	1937511	**-6.10#**
TS	1901675	1983710	**-20.33#**	1890775	1968400	**-19.78#**	2277521	2366292	**-8.31#**	1790750	1876627	**-22.59#**	2203700	2274656	**-28.48#**
instance	j303_1			j303_2			j303_3			j303_4			j303_5		
ACO	2594625	2705114		2599150	2644681		2562625	2680783		2217775	2292801		2455692	2662738	
ACO-L	**2350800**	**2455833**		**2364175**	**2462287**		**2402650**	**2505496**		**2136625**	**2191371**		**2395000**	**2487944**	
3dGA	3154875	3317060	**-23.82#**	3135050	3262186	**-28.37#**	3617300	3758402	**-37.91#**	2901900	2978116	**-34.52#**	3122634	3277523	**-27.42#**
KGA	3528700	3666935	**-41.48#**	3405575	3565378	**-37.44#**	3508725	3753438	**-37.22#**	2741075	2857078	**-28.19#**	3458175	3628828	**-31.84#**
KGA-p	2629600	2794833	-1.74	2613600	2742433	**-4.27#**	2581475	2758113	**-3.76#**	2259300	2354277	**-2.56#**	2524992	2725462	**-4.58#**
TS	3060950	3166678	**-20.87#**	3023725	3146018	**-21.01#**	3103250	3254565	**-28.29#**	2498950	2561257	**-13.44#**	3082142	3195475	**-20.46#**
instance	j303_6			j303_7			j303_8			j303_9			j303_10		
ACO	2624650	2729980		2569800	2634844		2459200	2520150		2401550	2469718		2081000	2131666	
ACO-L	**2558475**	**2656724**		**2243675**	**2324251**		**2260600**	**2365185**		**2189350**	**2270084**		**2017925**	**2073903**	
3dGA	3582973	3718380	**-44.74#**	2928700	3153626	**-18.99#**	2915125	3074436	**-20.69#**	2972375	3102171	**-28.11#**	2749875	2892953	**-18.53#**
KGA	3543525	3651106	**-41.39#**	2927325	3141253	**-18.82#**	3107350	3240867	**-33.62#**	3155138	3266871	**-35.00#**	2898125	2995122	**-35.07#**
KGA-p	2820725	2933986	**-14.33#**	2403647	2555134	6.25*	2492975	2604188	**-3.85#**	2418400	2542894	**-5.16#**	2129325	2267877	**-2.41#**
TS	3016625	3171516	**-23.03#**	2745616	2809714	**-9.88#**	2814900	2866416	**-11.47#**	2759625	2857524	**-14.06#**	2496100	2605753	**-12.20#**

comprehensive, we also report the results of proposed ACO approach without using the local mutation procedure presented in Section 5.4. We denote the ACO approach without the local mutation procedure as "ACO" in the table, and the one with the local mutation procedure is denoted as "ACO-L."

According to the results, it can be seen that out of the 83 instances, the ACO-L with the local mutation procedure yields the best results and the best average results in all cases. In some instances, the ACO-L can reduce the costs by 10-20 percent compared with other approaches. Even without the local mutation procedure, the proposed ACO approach

TABLE 2
(Cont.)

	best	mean	t-test	best	mean	t-test	best	mean	t-test	best	mean	t-test	best	mean	t-test
instance	j601_1			j601_2			j601_3			j601_4			j601_5		
ACO	3695625	3792208		3046950	3285752		2424375	2601930		3192875	3263161		3568550	3919305	
ACO-L	**3158950**	**3445884**		**2589400**	**2823333**		**2275550**	**2460015**		**2700475**	**2817612**		**3189700**	**3338418**	
3dGA	4065500	4296090	**-23.94#**	3541900	3812393	**-18.70#**	2814600	2944684	**-10.23#**	3405700	3623943	**-14.33#**	4977520	5157157	**-39.25#**
KGA	4562600	4695587	**-35.30#**	3993000	4196070	**-25.24#**	3679300	3818952	**-41.74#**	3958450	4134023	**-25.28#**	5284223	5601948	**-51.70#**
KGA-p	3683250	3885065	-1.67	3144750	3352059	**-3.16#**	2608800	2839034	**-8.97#**	3166700	3380224	**-4.86#**	3697200	3913237	1.28
TS	4109875	4210402	**-20.52#**	3557800	3648776	**-9.76#**	3138575	3270767	**-24.20#**	3506100	3596940	**-10.00#**	4628394	4754391	**-31.11#**
instance	j601_6			j601_7			j601_8			j601_9			j601_10		
ACO	3168875	3256873		3455995	3758105		3172458	3337052		4067200	4189292		4135228	4344357	
ACO-L	**2593100**	**2833035**		**3098875**	**3277243**		**2879400**	**3069308**		**3475470**	**3631250**		**3420711**	**3739155**	
3dGA	3772075	4164701	**-21.01#**	4476626	4657161	**-29.79#**	4425150	4651939	**-28.44#**	4766850	4932598	**-13.04#**	5422135	5654670	**-27.73#**
KGA	3869450	3987908	**-17.38#**	5070225	5245393	**-34.24#**	4232375	4384964	**-24.13#**	5259865	5429825	**-24.12#**	4819805	4978656	**-14.78#**
KGA-p	3237625	3577624	**-10.18#**	3549094	3826623	**-2.23#**	3255950	3532113	**-4.12#**	3854845	4039336	6.84*	3951705	4260009	9.38*
TS	3387950	3490600	**-8.30#**	4395335	4548620	**-26.48#**	3777375	3910975	**-8.76#**	4700300	4834199	**-13.03#**	4354025	4507374	**-7.05#**
instance	j602_1			j602_2			j602_3			j602_4			j602_5		
ACO	3178850	3290725		3131850	3211442		2670450	2762487		3577238	3663443		4072912	4266200	
ACO-L	**2841225**	**2961262**		**2714775**	**2835058**		**2217275**	**2382618**		**3147900**	**3270267**		**3386325**	**3516753**	
3dGA	4037450	4279513	**-31.72#**	3797025	4020995	**-23.91#**	2895125	3098081	**-11.42#**	4516950	4825491	**-30.90#**	5253950	5486705	**-23.74#**
KGA	4982450	5187719	**-55.80#**	4094893	4327598	**-31.66#**	3697700	3902481	**-31.09#**	4663077	4834979	**-33.19#**	5487275	5683303	**-28.73#**
KGA-p	3609075	3927764	**-20.94#**	3126575	3421118	**-8.64#**	2805500	3029793	**-9.57#**	3747442	4040265	**-14.67#**	4084050	4376127	**-4.92#**
TS	4030400	4206843	**-29.85#**	3681500	3798560	**-15.55#**	3106000	3228101	**-13.37#**	4019740	4123738	**-16.08#**	4680350	4825580	**-13.48#**
instance	j602_6			j602_7			j602_8			j602_9			j602_10		
ACO	3300177	3502046		3385350	3482738		3952450	4071442		2970125	3068901		4230179	4415250	
ACO-L	**2997550**	**3139477**		**2955825**	**3067753**		**3323046**	**3491968**		**2457125**	**2608666**		**3775175**	**3986992**	
3dGA	4353575	4533541	**-22.47#**	3666975	3907796	**-11.90#**	5074975	5386065	**-29.06#**	3583400	3788958	**-14.82#**	6086550	6329343	**-43.29#**
KGA	5100350	5289568	**-42.97#**	4377150	4570035	**-29.98#**	5068533	5248048	**-25.28#**	4002400	4180343	**-39.96#**	5617275	5825924	**-26.41#**
KGA-p	3377900	3606182	**-4.47#**	3407325	3666551	**-7.52#**	3855025	4195027	**-4.20#**	2846350	3183487	**-4.087#**	4699650	4971980	**-11.24#**
TS	4317775	4467125	**-20.01#**	3848400	3985298	**-12.26#**	4429900	4544814	**-10.01#**	3483175	3568639	**-10.01#**	4996800	5131117	**-15.90#**
instance	j603_1			j603_2			j603_3			j603_4			j603_5		
ACO	5091700	5196168		5070100	5222007		4163250	4140128		4902125	5100516		5787275	6047794	
ACO-L	**4269125**	**4423030**		**4182075**	**4391673**		**3803500**	**3973106**		**4512100**	**4596388**		**5291875**	**5590910**	
3dGA	6161050	6324694	**-26.40#**	5590250	7021234	**-40.20#**	4901225	5512341	**-23.89#**	6012475	7012281	**-22.09#**	6912081	8019980	**-30.42#**
KGA	6034875	6217798	**-21.23#**	6277300	6471582	**-24.78#**	5493275	5716510	**-26.49#**	6435975	6675438	**-18.38#**	7514150	7736445	**-22.08#**
KGA-p	4745525	5020547	8.21*	4788475	5084406	7.33*	4197864	4461402	**-4.42#**	5011575	5160186	**-2.21#**	5828975	6153543	-1.96
TS	5317325	5457419	**-9.76#**	5489900	5601935	**-11.20#**	4825100	4981622	**-11.11#**	5525475	5798739	**-9.71#**	6777275	6936233	**-15.40#**
instance	j603_6			j603_7			j603_8			j603_9			j603_10		
ACO	5312987	5436305		3870293	3938450		4628499	4901242		4102049	4312480		5392958	5529858	
ACO-L	**4634150**	**4774527**		**3452500**	**3625907**		**4029167**	**4294788**		**3834025**	**3977598**		**4934750**	**5174285**	
3dGA	5810950	6581090	**-23.88#**	4024850	5124801	**-22.38#**	5810941	6629840	**-50.82#**	4219025	5024915	**-7.68#**	6000250	6720248	**-14.29#**
KGA	6856445	7037984	**-31.40#**	5716386	5910279	**-37.50#**	5858634	6100522	**-40.35#**	5861175	6105670	**-24.36#**	6750050	6977748	**-20.08#**
KGA-p	5207424	5457459	-1.67	4005150	4287361	**-9.17#**	4591342	4859928	3.29*	4274175	4528618	**-2.91#**	5580225	5856382	**-6.17#**
TS	6002475	6142526	**-16.78#**	4901400	5067660	**-10.86#**	5242892	5343090	**-27.61#**	5042400	5239876	**-12.23#**	5977025	6131786	**-9.97#**
instance	j901_1			j901_2			j901_3			j901_4			j901_5		
ACO	10048392	10226310		9986400	10147857		8762322	9003288		11331840	11608409		8928484	9029428	
ACO-L	**8875497**	**9524273**		**8482017**	**9065699**		**7590922**	**7942599**		**9473400**	**9979136**		**8252640**	**8582510**	
3dGA	14461900	15084642	**-41.76#**	14902800	15421810	**-37.61#**	11403900	11636791	**-19.70#**	14466080	14869357	**-44.22#**	12392040	12843953	**-30.91#**
KGA	11213030	11542145	**-11.30#**	11685120	12080865	**-14.20#**	10945826	11208572	**-16.44#**	12293405	12551805	**-12.47#**	10600680	10835399	**-16.15#**
KGA-p	10193060	11170273	**-6.11#**	9741365	10689363	**-3.34#**	8802898	9132758	**-2.67#**	10714780	11288049	5.27*	8819560	9592516	**-4.99#**
TS	10138742	10290838	0.39	10483720	10690707	**-3.72#**	9568600	9816713	**-3.88#**	11029026	11256746	5.50*	9534666	9728068	**-8.96#**
instance	j901_6			j901_7			j901_8			j901_9			j901_10		
ACO	1033546	10916237		11438428	12123113		8357091	8870238		10230294	10325742		9487289	10103208	
ACO-L	**9515980**	**10034701**		**10156871**	**10740634**		**7585800**	**8096608**		**8598080**	**9346379**		**8031260**	**8429413**	
3dGA	14948380	15368398	**-31.07#**	15888120	16720656	**-41.01#**	12150800	12587776	**-39.49#**	14427640	14772240	**-40.29#**	13204640	13825023	**-36.65#**
KGA	11935380	12404230	**-14.82#**	12141060	12606255	**-7.92#**	10942000	11242554	**-25.40#**	11142720	11452110	**-17.33#**	11205460	11604536	**-19.96#**
KGA-p	10550220	11462967	**-7.37#**	11412540	12155832	-0.38	8637000	9265232	**-5.91#**	10188450	11080771	**-5.67#**	9361360	10118043	**-2.02#**
TS	10901300	11139570	**-3.26#**	11106740	11316258	8.92*	9583720	9781689	**-12.48#**	10052520	10242012	3.52*	9903920	10112942	-1.74

#The results of ACO are significantly better according to a t-test at level 0.05.
*The results of ACO are significantly worse according to a t-test at level 0.05.

still has competitive performance. We implement t-tests to find out if there are significant differences between the results found by the ACO without the local mutation procedure and those found by other algorithms. The results of t-tests are also reported in Table 2. According to these results, the ACO without local search still significantly outperforms the 3dGA on 82 instances, the KGA on all 83 instances, the TS on 77 instances, and the KGA-p on 56 instances. Overall, these results show that ACO is able to obtain project plans with lower costs in most instances.

We further analyze the convergence behavior of the algorithms based on their evolutionary curves. The evolutionary curves which give the best results found by the algorithms (averaged over 30 runs) as a function of the

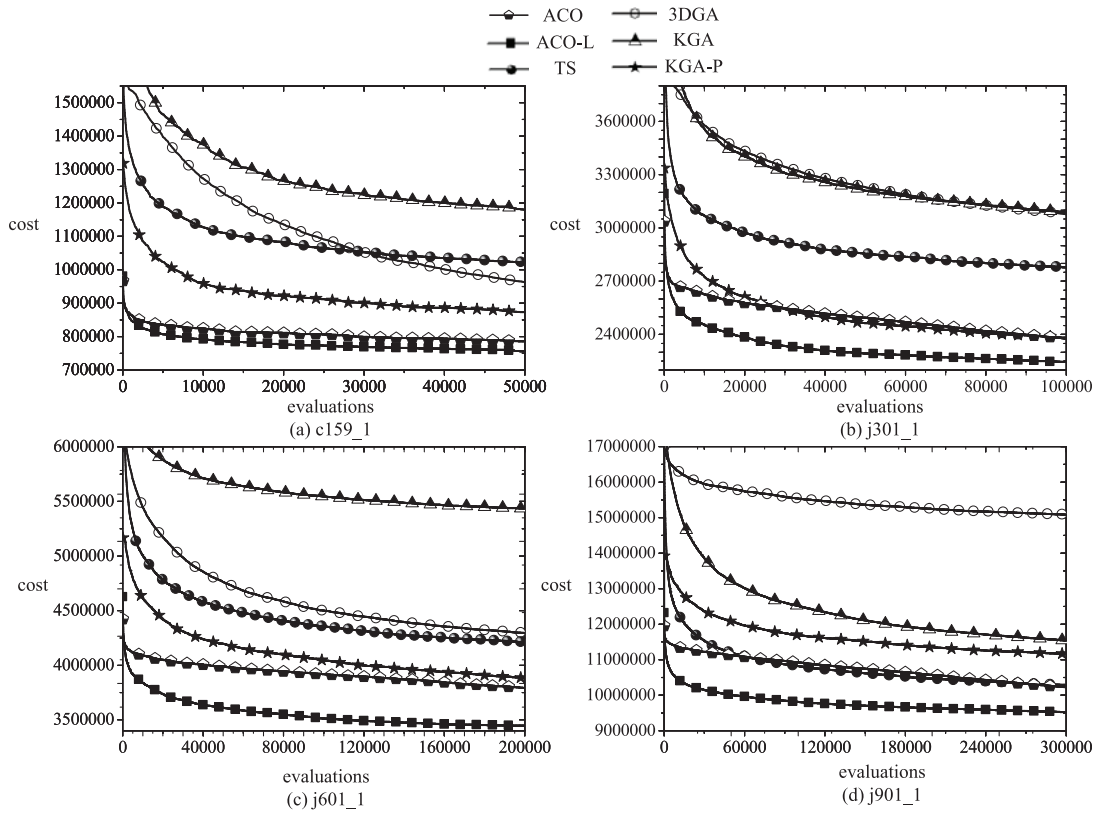

Fig. 6. Comparison of the evolutionary curves of different algorithms.

number of evaluations are illustrated in Fig. 6. The plots show that ACO can usually find good solutions at the very beginning of the search process and maintain its advantage till the end of the search process. This is because the proposed ACO adopts some useful heuristic information to guide the construction process and uses the aggressive pseudorandom proportional selection rule of ACS [34] in the solution construction procedure. With the help of the local mutation procedure, the ACO-L can further improve the performance of the ACO approach.

Finally, it is noteworthy from Table 2 and Fig. 6 that the KGA-p outperforms the KGA on most instances. These results demonstrate that the proposed EBS is effective. By coupling with the EBS, the KGA-p is capable of dealing with task preemption. Thus, it provides a more flexible and efficient way for human resource allocation and its performance is improved. On the other hand, while both the proposed ACO and the KGA-p are coupled with the EBS, ACO can still achieve better results than the KGA-p in most cases. These results also demonstrate that the proposed ACO algorithm is effective.

6.3 Comparison of the Plan Structures

To further analyze why the proposed approach can yield good results, we further go deep into the structures of the solutions found by different algorithms. We plot a section of the Gantt charts of the plans found by the ACO, 3dGA, and TS in Fig. 7.

The 3dGA proposed in [6] allows task preemption and schedules the workloads period by period based on the 3D employee allocation matrix. Because the workloads are assigned month by month separately, it is possible that a group of employees is assigned to a task in a month and another different group of employees is assigned to the task in the next month. For example in Fig. 7b, the task 6 is executed by employees #2 and #6 on May and by employees #1 and #5 on June. Such a plan is actually unstable and impractical for real-world software projects.

On the other hand, the TS [19] (and also the KGA [18]) is designed for the multiskill scheduling problem. The models of these studies do not allow task preemption and usually assume that an employee can only be assigned to a task at one time. This assumption reduces the flexibility of human resource allocation and may sometimes lead to inefficient plans. For example, in the part of a plan built by the TS given in Fig. 7c, as employee #1 is assigned to work for task 3 from March to May, the execution of task 2 has to be delayed until employee #1 is finally released from task 3 in June.

Compared with the above approaches, with the EBS, the plans generated by the proposed approach seem to be more flexible and practical. For example, in Fig. 7a the workload assignments of employees are generally stable because the assigned working hours of employees are all adjusted based on the same planned employee allocation matrix. Moreover, when a task is finished and its related recourses are released, these resources can be assign to other tasks immediately (e.g., employee #1 joins tasks 3 and 6 in May after he finished his work on task 4 in March and April). In this way, the human resources can be utilized in a more efficient manner.

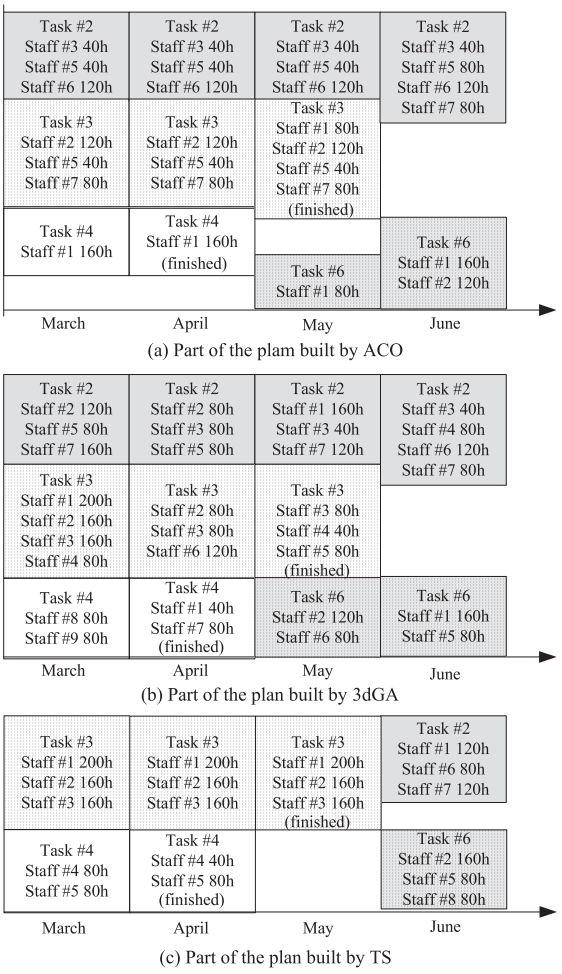

Fig. 7. Comparison between the structures of the solutions built by different algorithms.

6.4 Analysis of the Local Refinement Steps

Finally, we analyze the functions of the two local refinement steps in the proposed EBS. The first local refinement step in the line 17 of Fig. 2 is to let regular employees dedicate all of their normal working hours to the project, and the second local refinement step in the line 24 is to release redundant assignments when a task is complete. Fig. 8 illustrates the performance of these local refinement steps on the instance j301_1.

According to the figure, both the schemes of only using local refinement step 1 (scheme "local 1") and only using local refinement step 2 (scheme "local 2") yield better results than the scheme without any local refinement step (scheme "without local"). These results prove that both refinement steps are helpful. Therein, local refinement step 2 is more significant to the performance of the algorithm as the scheme "local 2" yields better results than the scheme "local 1." Nevertheless, by using both the local refinement steps, the proposed method manages to obtain the best results among the four schemes shown in Fig. 7.

7 Conclusion

A new method for solving the software project planning problem has been developed. The main characteristics of the proposed method are in two aspects. First, the method introduces an event-based scheduler. Second, the method takes advantage of ACO to solve the complicated planning problem. Experimental results show that the representation scheme with the EBS is effective, and the proposed algorithm manages to yield better plans with lower costs and more stable workload assignments compared with other existing approaches.

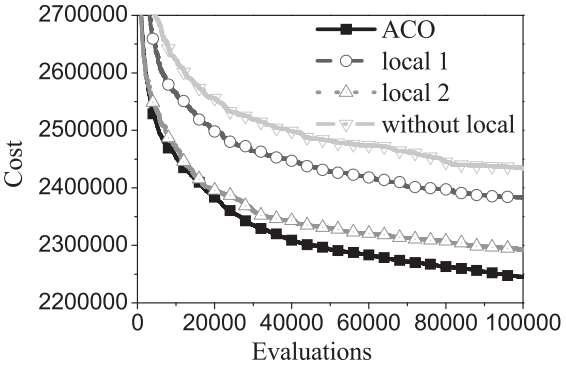

Fig. 8. Analysis of the local refinement steps on the instance j301_1.

In future research, it will be interesting to consider employee experience and the training model [6] to make the considered problem more comprehensive. Including uncertainty treatment in the software project planning model is also a promising research topic. In addition, since the model proposed in this paper provides a flexible and effective way for managing human resources, it is promising to apply the proposed approach to other complex human-centric projects like consulting projects.

Acknowledgments

This work was supported in part by the National Science Fund for Distinguished Young Scholars under Grant 61125205, in part by the National natural Science Foundation of China under Grant 61070004, and in part by the NSFC Joint Fund with Guangdong under Key Project U0835002.

References

[1] N. Nan and D.E. Harter, "Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort," *IEEE Trans. Software Eng.*, vol. 35, no. 5, pp. 624-637, Sept./Oct. 2009.
[2] J. Duggan, H. Byrne, and G.J. Lyons, "A Task Allocation Optimizer for Software Construction," *IEEE Software*, vol. 21, no. 3, pp. 76-82, May/June 2004.
[3] R.-G. Ding and X.-H. Jing, "Five Principles of Project Management in Software Companies," *Project Management Technology* (in Chinese), vol. 1, 2003.
[4] A. Barreto, M. de O. Barros, C.M.L. Werner, "Staffing a Software Project: A Constraint Satisfaction and Optimization-Based approach," *Computers & Operations Research*, vol. 35, pp. 3073-3089, 2008.
[5] E. Alba and J.F. Chicano, "Software Project Management with GAs," *Information Sciences*, vol. 177, pp. 2380-2401, 2007.
[6] C.K. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-Line Based Model for Software Project Scheduling with Genetic Algorithms," *Information and Software Technology*, vol. 50, pp. 1142-1154, 2008.
[7] B. Boehm, *Software Engineering Economics.* Prentice-Hall, 1981.
[8] B. Boehm et al., *Software Cost Estimation with COCOMO II.* Prentice-Hall, 2000.

[9] A. Shtub, J.F. Bard, and S. Globerson, *Project Management: Processes, Methodologies, and Economics,* second ed. Prentice Hall, 2005.

[10] P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch, "Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods," *European J. Operational Research,* vol. 112, pp. 3-41, 1999.

[11] C.K. Chang and M. Christensen, "A Net Practice for Software Project Management," *IEEE Software,* vol. 16, no. 6, pp. 80-88, Nov./Dec. 1999.

[12] C.K. Chang, M.J. Christensen, C. Chao, and T.T. Nguyen, "Software Project Management Net: A New Methodology on Software Management," *Proc. 22nd Ann. Int'l Computer Software and Applications Conf.,* 1998.

[13] A. Kumar V.K. and L.S. Ganesh, "Use of Petri Nets for Resource Allocation in Projects," *IEEE Trans. Eng. Management,* vol. 45, no. 1, pp. 49-56, Feb. 1998.

[14] L.C. Liu and E. horowitz, "A Formal Model for Software Project Management," *IEEE Trans. Software Eng.,* vol. 15, no. 10, pp. 1280-1293, Oct. 1989.

[15] C.K. Chang, M.J. Christensen, and T. Zhang, "Genetic Algorithms for Project Management," *Annals of Software Eng.,* vol. 11, pp. 107-139, 2001.

[16] L.-H. Lee, "Robust Preemptive Resource Assignment for Multiple Software Projects Using Parameter Design," *Int'l J. Applied Science and Eng.,* vol. 5, no. 2, pp. 159-171, 2007.

[17] J. Wglarz, J. Józefowska, M. Mika, and G. Waligóra, "Project Scheduling with Finite or Infinite Number of Activity Processing Modes—A Survey," *European J. Operational Research,* vol. 208, pp. 177-205, 2011.

[18] V. Yannibelli and A. Amandi, "A Knowledge-Based Evolutionary Assistant to Software Development Project Scheduling," *Expert Systems with Applications,* vol. 38, pp. 8403-8413, 2011.

[19] O. Bellenguez and E. Néron, "Methods for the Multi-Skill Project Scheduling Problem," *Proc. Ninth Int'l Workshop Project Management and Scheduling,* pp. 66-69, 2004.

[20] O. Bellenguez and E. Néron, "A Branch-and-Bound Method for Solving Multi-Skill Project Scheduling Problem," *RAIRO-Operations Research,* vol. 41, no. 2, pp. 155-170, 2007.

[21] J. Clarke et al., "Reformulating Software Engineering as a Search Problem," *Proc. IEE Software,* vol. 150, no. 3, pp. 161-175, 2003.

[22] M. Harman, S.A. Mansouri, and Y. Zhang, "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications," Technical Report TR-09-03, Apr. 2009.

[23] J.S. Aguilar-Ruiz, I. Ramos, J.C. Riquelme, and M. Toro, "An Evolutionary Approach to Estimating Software Development Projects," *Information and Software Technology,* vol. 43, pp. 875-882, 2001.

[24] K. Praditwong, M. Harman, and X. Yao, "Software Module Clustering as a Multi-Objective Search Problem," *IEEE Trans. Software Eng.,* vol. 37, no. 2, pp. 264-282, Mar./Apr. 2011.

[25] C.L. Simons, I.C. Parmee, and R. Gwynllyw, "Interactive, Evolutionary Search in Upstream Object-Oriented Class Design," *IEEE Trans. Software Eng.,* vol. 36, no. 6, pp. 798-816, Nov./Dec. 2010.

[26] M. Harman and P. McMinn, "A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search," *IEEE Trans. Software Eng.,* vol. 36, no. 2, pp. 226-247, Mar./Apr. 2010.

[27] V. Garousi, "A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation," *IEEE Trans. Software Eng.,* vol. 36, no. 6, pp. 778-797, Nov./Dec. 2010.

[28] A. Ngo-The and G. Ruhe, "Optimized Resource Allocation for Software Release Planning," *IEEE Trans. Software Eng.,* vol. 35, no. 1, pp. 109-123, Jan./Feb. 2009.

[29] E. Alba and J.F. Chicano, "Management of Software Projects with GAs," *Proc. Sixth Mateheuristics Int'l Conf.,* P. 1152, 2005.

[30] G. Antoniol, M. Di Penta, and M. Harman, "Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project," *Proc. 21st IEEE Int'l Conf. Software Maintenance,* 2005.

[31] G. Antoniol, M. Di Penta, and M. Harman, "Search-Based Techniques for Optimizing Software Project Resource Allocation," *Genetic and Evolutionary Computation,* vol. 3103, pp. 1425-1436, 2004.

[32] Y. Ge, "Software Project Rescheduling with Genetic Algorithms," *Proc. Int'l Conf. Artificial Intelligence and Computational Intelligence,* 2009.

[33] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Trans. Systems Man, and Cybernetics-Part B: Cybernetics,* vol. 26, no. 1, pp. 29-41, Feb. 1996.

[34] M. Dorigo and L.M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to TSP," *IEEE Trans. Evolutionary Computation,* vol. 1, no. 1, pp. 53-66, Apr. 1997.

[35] D. Merkle, M. Middendorf, and H. Schmeck, "Ant Colony Optimization for Resource-Constrained Project Scheduling," *IEEE Trans. Evolutionary Computation,* vol. 6, no. 4, pp. 333-346, Aug. 2002.

[36] W.-N. Chen and J. Zhang, "An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem with Various QoS Requirements," *IEEE Trans. System, Man, and Cybernetics-Part C,* vol. 39, no. 1, pp. 29-43, Jan. 2009.

[37] W.-N. Chen, J. Zhang, H.S.-H. Chung, R.-Z. Huang, and O. Liu, "Optimizing Discounted Cash Flows in Project Scheduling—An Ant Colony Optimization Approach," *IEEE Trans. Systems, Man, and Cybernetics-Part C,* vol. 40, no. 1, pp. 64-77, Jan. 2010.

[38] X.-M. Hu, J. Zhang, H.S.-H. Chung, O. Liu, and J. Xiao, "An Intelligent Testing System Embedded with an Ant Colony Optimization Based Test Composition Method," *IEEE Trans. Systems, Man and Cybernetics, Part C: Applications and Rev.,* vol. 39, no. 6, pp. 659-669, Nov. 2009.

[39] M. Dorigo and T. Stützle, *Ant Colony Optimization.* MIT Press, 2004.

[40] L. Özdamar, "A Genetic Algorithm Approach to a General Category Project Scheduling Problem," *IEEE Trans. Systems, Man, and Cybernetics-Part C: Applications and Rev.,* vol. 29, no. 1, pp. 44-59, Feb. 1999.

[41] R. Bai, E.K. Burke, G. Kendall, J. Li, and B. McCollum, "A Hybrid Evolutionary Approach to the Nurse Rostering Problem," *IEEE Trans. Evolutionary Computation,* vol. 14, no. 4, pp. 580-590, Aug. 2010.

[42] V. Nissen and M. Günther, "Automatic Generation of Optimised Working Time Models in Personnel Planning," *Proc. Seventh Int'l Conf. Swarm Intelligence,* pp. 384-391, 2010.

[43] M. Günther and V. Nissen, "Particle Swarm Optimization and an Agent-Based Algorithm for a Problem of Staffing Scheduling," *Proc. Int'l Conf. Applications of Evolutionary Computation,* pp. 451-461, 2010.

[44] T. Stützle and H. Hoos, "Max-Min Ant System," *Future Generation Computer Systems,* vol. 16, no. 8, pp. 889-914, 2000.

[45] A. Bauer, B. Bullnheimer, R.F. Hartl, and C. Strauss, "Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization," *Central European J. Operations Research and Economics,* vol. 8, no. 2, pp. 125-141, 2000.

[46] B. Pfahringer, "Multi-Agent Search for Open Shop Scheduling: Adapting the Ant-Q Formalism," Technical Report TR-96-09, Austrian Research Inst. of Artificial Intelligence, 1996.

[47] C. Blum and M. Sampels, "An Ant Colony Optimization Algorithm for Shop Scheduling Problems," *J. Math. Modelling and Algorithms,* vol. 3, pp. 285-308, 2004.

[48] C. Blum, "Beam-ACO-Hybridizing Ant Colony Optimization with Beam Search: An Application to Open Shop Scheduling," *Computers and Operations Research,* vol. 32, pp. 1565-1591, 2005.

[49] R. Kolisch and S. Hartmann, "Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis," *Handbook on Recent Advances in Project Scheduling,* J. Weglarz, ed., pp. 197-212, Kluwer, 1999.

[50] R. Kolisch and A. Sprecher, "PSPLIB—A Project Scheduling Problem Library: OR Software—ORSEP Operations Research Software Exchange Program," *European J. Operational Research,* vol. 96, no. 1, pp. 205-216, 1997.

[51] L. Özdamar and G. Ususoy, "A Survey on the Resource-Constrained Project Scheduling Problem," *Trans. IIE,* vol. 27, pp. 574-586, 1995.

Wei-Neng Chen received the bachelor's degree in network engineering and the PhD degree in tcompuuter application technology from the Sun Yat-sen University, Guangzhou, China, in 2006 and 2012, respectively. He is currently a lecturer with the Department of Computer Science, Sun Yat-sen University, China. His research interests include evolutionary computation and its applications on financial optimization, operations research, and software engineering. He is a member of the IEEE.

Jun Zhang received the PhD degree from the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, China, in 2002. He is currently working as a professor with the Department of Computer Science, Sun Yat-Sen University, Guangzhou, China. He has authored seven research books and book chapters, and more than 100 technical papers in his research areas. His research interests include computational intelligence, operations research, data mining, wireless sensor networks, and power electronic circuits. He received the National Science Fund for Distinguished Young Scholars in 2011. He received the First-Grade Award in Science Research from the Ministry of Education, China, in 2009. He is currently an associate editor for the *IEEE Transactions on Industrial Electronics*, the *IEEE Transactions on Evolutionary Computation*, and the *IEEE Computational Intelligence Magazine*. He is the founding and current chair of the IEEE Guangzhou Subsection and the IEEE Beijing (Guangzhou) Section Computational Intelligence Society Chapter. He is a senior member of the IEEE.