Dynamic Software Project Scheduling through a Proactive-rescheduling Method

Shen, Xiao-ning; Minku, Leandro; Bahsoon, Rami; Yao, Xin

DOI:
10.1109/TSE.2015.2512266

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Shen, X, Minku, L, Bahsoon, R & Yao, X 2016, 'Dynamic Software Project Scheduling through a Proactive-rescheduling Method', IEEE Transactions on Software Engineering, vol. 42, no. 7, pp. 658 - 686. https://doi.org/10.1109/TSE.2015.2512266

Dynamic Software Project Scheduling through a Proactive-Rescheduling Method

Xiaoning Shen, Leandro L. Minku, *Member, IEEE*, Rami Bahsoon, and Xin Yao, *Fellow, IEEE*

Abstract—Software project scheduling in dynamic and uncertain environments is of significant importance to real-world software development. Yet most studies schedule software projects by considering static and deterministic scenarios only, which may cause performance deterioration or even infeasibility when facing disruptions. In order to capture more dynamic features of software project scheduling than the previous work, this paper formulates the project scheduling problem by considering uncertainties and dynamic events that often occur during software project development, and constructs a mathematical model for the resulting multi-objective dynamic project scheduling problem (MODPSP), where the four objectives of project cost, duration, robustness and stability are considered simultaneously under a variety of practical constraints. In order to solve MODPSP appropriately, a multi-objective evolutionary algorithm based proactive-rescheduling method is proposed, which generates a robust schedule predictively and adapts the previous schedule in response to critical dynamic events during the project execution. Extensive experimental results on 21 problem instances, including three instances derived from real-world software projects, show that our novel method is very effective. By introducing the robustness and stability objectives, and incorporating the dynamic optimization strategies specifically designed for MODPSP, our proactive-rescheduling method achieves a very good overall performance in a dynamic environment.

Index Terms—Schedule and organizational issues, dynamic software project scheduling, search-based software engineering, multi-objective evolutionary algorithms, mathematical modeling

1 Introduction

Effective software project scheduling is crucial, when managing the development of medium to large scale projects to meet the deadline and budget [1]. The process of software project scheduling includes some duties [1], [2]: "identify project activities; identify activity dependencies; estimate resources for activities; allocate people to activities; and create project charts." The so-called project scheduling problem (PSP) [2], [3], [4], [5] deals with the fourth duty which allocates employees with certain skills to activities (tasks) so that the required objectives (project cost, duration, etc.) can be achieved subject to various constraints. Good allocations are very important for software projects, since human resources are their main resources [6]. PSP is solved based on the information obtained from prior duties, i.e., the indentified tasks, task dependencies, and the estimated effort required for tasks provided by the software manager. Besides, information about the available employees and their salaries and skills is also needed. PSP has been tackled by both classical and meta-heuristic approaches. The classical methods include the program evaluation and review technique [7] and the critical path method [8], which represent projects by activity-on-the-arc networks, and the resource-constrained project scheduling problem model [9]. PSP has also been formulated as a search-based optimization problem in [10], [11], [12] to provide near-optimal schedules in a large search space, and to automate the task of allocations, which would otherwise be performed by humans [2].

In previous studies on software project scheduling, it was assumed that the system information, such as the effort required by each task and the skills of each employee, are known beforehand and remain unchanged. They also assumed that no disruptions occur during the project lifetime to interrupt the task execution. However, in the real world, the working environment changes dynamically [1] by unpredictable events, such as requirement changes during the lifecycle of a project, a new urgent task arriving suddenly, an employee leaving, etc. A previously optimal schedule may become obsolete and infeasible in the new environment. Moreover, it is common that project activities are subject to considerable uncertainties. For instance, the task effort may have been estimated incorrectly, the task specification may be modified so that the originally estimated effort required by the task is changed, the employee skill level may be improved because of increasing experience, etc. The optimal schedule generated according to the initial data may have large performance deterioration when facing disturbances. Pressman [14] indicated eight reasons for late software delivery, five of which are related to uncertainties, risks and unpredictable events appearing during the project execution, which are: "changing customer requirements that are not reflected in schedule changes; an honest underestimate of the

- *X. Shen is with B-DAT & CICAEET, School of Information and Control, Nanjing University of Information Science and Technology, No.219, Ning-Liu Road, Pu-Kou District, Nanjing 210044, P.R. China. E-mail: sxnystsyt@sina.com.*
- *L.L. Minku is with the Department of Computer Science, University of Leicester, University Road, Leicester LE17RH, United Kingdom. E-mail: leandro.minku@leicester.ac.uk.*
- *R. Bahsoon and X. Yao are with CERCIA, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom. E-mail: r.bahsoon@cs.bham.ac.uk., X.Yao@cs.bham.ac.uk.*

Manuscript received 20 Mar. 2014; revised 9 July 2015; accepted 15 Dec. 2015. Date of publication 23 Dec. 2015; date of current version 22 July 2016. Recommended for acceptance by M. Cohen.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2015.2512266

amount of effort and/or the number of resources that will be required to do the job; predictable and/or unpredictable risks that were not considered when the project commenced; technical difficulties that could not have been foreseen in advance; and human difficulties that could not have been foreseen in advance." Thus, it is vital to develop a dynamic software project scheduling approach which can deal with both uncertainties and dynamic events to reduce the late software delivery. Furthermore, software engineering in emerging paradigms (e.g. the cloud, mobility, ultra-large software systems) calls for new scheduling methods that explicitly cater for uncertainties and dynamism in scheduling. This is because many of the requirements may be unique to the said project and exhibit little resemblance to prior projects. Consequently, static scheduling methods may be ineffective and may render myopic outcome if used. In the field of scheduling, there are mainly three approaches to dynamic scheduling: completely reactive, predictive-reactive, and proactive (robust) scheduling [15]. Completely reactive scheduling creates partial schedules for the immediate future based on local information at each decision point. For example, when a machine becomes idle, the job with the highest priority will be selected from the waiting queue according to a priority dispatching rule. This approach is in essence a greedy one and can be trapped into a local optimum easily. Predictive-reactive scheduling has a scheduling/rescheduling process where previous schedules are adapted to the new environment caused by dynamic events, while proactive scheduling attempts to generate a schedule in advance, which has the ability to satisfy performance requirements predictably in an uncertain environment [16].

Although scheduling in dynamic and uncertain environments has attracted attention in construction and manufacturing domains [17], little effort has been made to capture the dynamic features of real-world software projects, let along multi-objective dynamic project scheduling problems (MODPSP). This paper tackles the challenge by first proposing a mathematical model to define the problem and then proposing a new proactive-rescheduling method that combines proactive and predictive-reactive scheduling to solve it. In static PSP, efficiency measures like project cost and duration are usually used as the objectives to be optimized. In dynamic PSP, a new schedule may be regenerated by simply minimizing the impact of disruptions to the project efficiency. For example, a software engineer may be redeployed on different tasks from the ones that he/she was originally assigned to. Consequently, he/she may need some time to learn and understand the newly assigned tasks, which delays the project, increases the cost/budget, and disrupts the smooth running of the project. To minimise potential negative impact of generating very different schedules in a dynamic environment, our MODPSP rescheduling process should create new schedules that differ as little as possible from the previous ones, i.e., it should promote stability in dynamic scheduling. Furthermore, given the existence of uncertainties in MODPSP, the schedule's quality should not be too sensitive to minor data variations, i.e., a good schedule should be robust against data variations. Therefore, MODPSP considers not only cost and duration as objectives, but also stability and robustness. Although there has been work on predictive scheduling for software projects under uncertainties [18], and on dynamic resource rescheduling in response to new project arrivals [19], there has not been any research work on the mathematical modeling and dynamic scheduling of MODPSP, which addresses both uncertainties and dynamic events occurring during the software project execution, as well as multi-objectivity under constraints.

The project cost, duration, robustness, and stability are usually conflicting with each other. It is useful to handle such multiple objectives using a true multi-objective approach, e.g., an multi-objective evolutionary algorithm (MOEA) [5], [11] that can provide various trade-offs among different objectives on the Pareto front. The Pareto front can help make informed decisions in dynamic scheduling.

The primary aim of this paper is to model the software project scheduling problem in a dynamic and uncertain environment by considering multiple objectives and constraints, and propose an MOEA-based proactive-rescheduling method for the formulated problem. Three aspects are studied: (i) PSP is formulated as a dynamic scheduling problem with one type of uncertainty and three kinds of dynamic events that often occur in software projects; (ii) the mathematical model for the MODPSP is constructed, considering the four objectives of project cost, duration, robustness and stability, and a variety of practical constraints; (iii) a proactive-rescheduling method is proposed to solve MODPSP. The key idea of the method is to create a robust schedule predictively considering the project uncertainties, and then revise the previous schedule by an MOEA-based rescheduling method in response to critical dynamic events. To evaluate the effectiveness of our method, 18 dynamic PSP benchmark instances and 3 instances derived from real-world software projects are used in our experimental studies, which have three major purposes: (1) investigating the influence of the robustness objective on proactive scheduling; (2) evaluating the strength and weakness of our MOEA-based rescheduling method over other dynamic scheduling methods which adjust the original schedule based on a simple heuristic rule; and (3) comparing the overall performance in dynamic environments obtained by five MOEA-based rescheduling methods, where the effectiveness of simultaneously considering project duration, cost, robustness and stability, and the dynamic optimization strategies adopted in our method are demonstrated.

This paper is organized as follows. Section 2 presents an overview of the related work. Section 3 describes our problem formulation and constructs the mathematical model of MODPSP. In Section 4, the framework of our proactive-rescheduling method is introduced, and the proposed rescheduling method called dε-MOEA is described. Section 5 details the techniques for individual representations, constraint handling and objective evaluations. Experimental analyses are presented in Section 6. Conclusions are drawn in Section 7.

2 Related Work

In PSP, there are a set of tasks and a group of employees. Each task has an effort expressed in person-month and a set of required skills. The tasks have to be carried out based on a task precedence graph (TPG), which specifies which tasks should finish before a new task starts. Each employee has a

salary and personal skills, a maximum degree of dedication to the project, and is able to do several tasks during a working day. PSP consists of determining which employees are allocated to each task and when each one should be performed, with the aim to minimize the project duration, minimize the project cost and so on, satisfying the constraints of task skills, no overwork, etc [3].

2.1 Software Project Scheduling with EAs in Static Environments

With the rapid development of search-based software engineering, there has been some work on software project scheduling based on EAs in the last decade. An early effort was from Chang et al. [4] who constructed a task-based model and applied a genetic algorithm (GA) to find near-optimal schedules. Alba and Chicano [3] used the same problem formulation as [4], and performed systematic empirical studies of the impact that important problem characteristics had on the solutions found by GAs. Also with such a problem formulation, Minku et al. [2] gave a runtime analysis to gain insight into how design choices in EAs affected performance on PSP, and which instances were easy or hard for EAs to solve.

To make their task-based model more practical, Chang et al. [12] presented a time-line model which split the task duration into small time units, and when evaluating the fitness of a solution, it assigned employees to tasks in discrete time units iteratively so that more human factors such as reassignment of employees, learning and training could be considered. However, this model introduced a lot of subjective parameters, to which the sensitivity of the solutions provided by the GA was unknown [2], and it would induce a large system instability because they scheduled tasks separately in different time units [10]. To preserve the flexibility in human resource allocation, Chen and Zhang [10] developed a model with an event-based scheduler which adjusted the allocations at events, and adopted an ant colony algorithm to solve the problem. Although the employee joining or leaving was considered as an event in [10], and the variations of human factors were allowed in [12], the software project scheduling was still treated as a static problem in these two studies, since it was assumed that when and how such events or variations occur were known in advance, which would be used for the fitness evaluation of each candidate solution. However, in the real-world software project, dynamic events or uncertainties usually occur in a stochastic way, and it is impossible to get all the accurate information in advance. Thus, it is more realistic to formulate the software project scheduling as a dynamic scheduling problem, and solve it dynamically during the project execution.

Luna et al. [5] and Chicano et al. [11] solved the static PSP by an MOEA based on Pareto domination [20], where cost and duration were not converted into a single combined function. Penta et al. [19] presented a comprehensive survey of the search-based techniques applied to software project scheduling and staffing.

2.2 Software Project Scheduling in Uncertain Environments

A few studies on software project scheduling under uncertainties have appeared recently. Hapke et al. [21] proposed a fuzzy software project scheduling system, where activity time parameters were uncertain and modeled by means of L-R fuzzy numbers, and the fuzzy problem was transformed into a set of associate deterministic problems. Lazarova-Molnar and Mizouni [22] gave a simulation based method to select the most appropriate remedial action scenario based on the project goal to limit the impact of uncertainties on the overall project success. Gueorguiev et al. [18] employed a proactive scheduling method where an MOEA was used to find the Pareto front which represented the trade-off between completion time and robustness (defined as the completion time difference when new tasks were added, or the tasks' durations were inflated). The work in [23] modeled the project scheduling using event chains. To obtain a schedule under uncertainties, a number of Monte Carlo simulations were performed based on a baseline project schedule and an event list. It can also be regarded as a proactive scheduling method. Antoniol et al. [24] used a tandem GA to find the best order for processing work packages and the best allocation of staff to project teams. Then a queuing simulator was used to analyze the sensitivity of the result obtained by GA with respect to uncertainties caused by effort estimation errors, reworks and abandonment on a given percentage of maintenance tasks. The result of this sensitivity analysis could guide the search which determined whether a negotiation of further people and a successive iteration of the tandem GA process were required. The whole process might repeat for multiple times to obtain a satisfied solution. The work in [24] just considered the robustness of the initial allocations to dynamic events of rework and abandonment, but not provided the responding strategies when they occurred. Chicano et al. [25] gave a new multi-objective formulation of PSP which considered the productivity of the employees in developing different tasks and the inaccuracies of task effort estimations. Task effort variations were assumed to follow the uniform distribution, and robustness was measured as the standard deviation of the make-span and cost values obtained from a certain number of simulations of task effort inaccuracies. In our work, both robustness to task effort uncertainties and immediate response to dynamic events are addressed by the proposed proactive-rescheduling method. Meanwhile, robustness is defined as the duration and cost increases from the initial values obtained in the case assuming no task effort uncertainties, where only the efficiency deterioration in the disrupted scenarios is penalized.

Xiao et al. [19] may be the first effort to consider dynamic resource rescheduling for addressing disruptions that happen during the software development. They used the little-JIL process definition language to describe the relations among different projects and project activities, where a project could be mapped into a task requiring a set of skills, and an activity could be mapped into one skill of a task in the task-based model proposed in [4]. There are three limitations in the work of [19]. Firstly, unlike the task-based model which searches the dedication degree of each employee to each task, Xiao et al. [19] just determined whether an employee should be allocated to each activity (skill) and the priority of each activity. The workload allocation of each employee to the assigned activity was not determined by the GA. Secondly, only one kind of disruptive

event which represented the introduction of a new project was considered, and rescheduling of merely three new project arrivals were conducted in their work. In practice, a variety of dynamic events may occur during the software development process. Moreover, continuous changes like task effort uncertainties widely exist, which indicates that the schedule robustness to uncertainties is also an important factor that should be taken into account. Thirdly, although the utility and process stability were considered in their work, they were converted into a single objective by a weighted sum method, introducing additional parameters in their objective definitions and weight determinations. Since multiple objectives are usually conflicting with each other, it is better to handle them by an MOEA which can provide various trade-offs among different objectives so that a project manager can make an informed decision when rescheduling.

In our work, we consider the dynamic version of task-based model, which determines the dedication of each employee to each task dynamically. The reason for using the task-based model is that it is more general. The work in [19] can be considered as a special case of the task-based model where each task requires a single skill. To address various uncertainties and real-time events, the task effort variances, employee leaves and returns, and new task (urgent or regular) arrivals are considered in our work. A clear mathematical model for the dynamic software project scheduling is developed, where four objectives, including the project cost, duration, robustness and stability, are considered simultaneously. An MOEA-based, proactive-rescheduling method is proposed to solve the dynamic scheduling problem.

3 Problem Formulation and Mathematical Modeling of MODPSP

3.1 Incorporating Dynamic Features into PSP

In order to address more dynamic characteristics of PSP, in this paper, one type of uncertainty and three kinds of dynamic events, which often occur during the execution of the real-world software project, are incorporated into PSP. They are listed as follows.

(1) *Task effort uncertainty.* At the beginning of the project, the effort required by each task can be estimated by some method such as the COCOMO model [26] or the more recent online learning model [27]. However, modifications in task specifications and inaccuracies in the initial estimations may cause the changes in the initially estimated task efforts. Here, task effort variances are assumed to follow a Normal distribution [3]. To infuse more reality, each task effort is assigned different values of mean and standard deviation. The mean value of each task is set to be its initially estimated task effort.

(2) *New task arrivals.* New requirements will emerge during the development lifecycle of software. This could be in response to changes in customers' requirements and/or the environment. It can also be attributed to the iterative and intertwined nature of the software development, where continuous refinements of requirements, architecture and designs can lead to new tasks. As the project progresses, the stakeholders' understanding of the project may evolve and new features may be added as a result. Furthermore, new requirements may also emerge as the software is prototyped, tested, or deployed. Such dynamism is very common in large and complex projects, where requirements tend to be highly "volatile" and changeable during the lifetime of the project. Consequently, the landscape of tasks tends to continuously evolve. Tasks can be classified into urgent and regular tasks. An urgent task should be performed immediately when it arrives, while a regular task does not have such a requirement. As volatility of requirements and its frequency are difficult to predict, we model the uncertainty of new task arrivals as following a Poisson distribution (i.e., the time between two new task arrivals is distributed exponentially).

(3) *Employee leaves.* Due to sickness or being part of multiple projects or other reasons, an employee may leave during the project. Here, employee leaves are assumed to follow a Poisson distribution. So for each employee, the time interval between leaves is assumed to follow an exponential distribution. To infuse more reality, each employee is assigned a different mean time between his/her leaves.

(4) *Employee returns.* After having been absent from the project, we consider that the employee may return back to the project. "employee returns" is the amount of time that the employee is absent from the project, i.e., the amount of time that passes from the moment the employee leaves until the employee returns to the project. Here, employee returns are also assumed to follow a Poisson distribution. To infuse more reality, each employee is assigned a different mean time to return, i.e., the time that an employee is out of the project.

It is worth noting that our approach is not limited to the Poisson distribution, which is often used in operations research. It is easy to replace the probability distribution used in our algorithm by any other appropriate probability distribution. All that is required is to plug in a different probability distribution to sample from.

Note that other types of uncertainties and dynamic events, such as changes in task precedence, addition of new employees not in the company before the project started, removal of tasks from the project due to changes in requirements, etc., may also occur during the dynamic process of a real-world project. As an illustration to validate the effectiveness and efficiency of the proposed proactive-rescheduling method, we only consider the task effort uncertainties, new task arrivals, employee leaves, and employee returns in the model of MODPSP and experimental studies used in this paper. The incorporation of other uncertainties and dynamic events is proposed as future work.

3.2 Employees' Properties

Assume a project requires a total of S skills and there are in total M employees involved in the project. Let t_l $(l = 0, 1, 2, \ldots)$ denote the scheduling point at which a rescheduling method is trigged (including the initial time

TABLE 1
Properties of Each Employee

name	description
e_i^{skills}	The skill indicator set of employee e_i. e_i^{skills}=$\{pro_i^1, pro_i^2, \ldots, pro_i^S\}$, where $pro_i^k \in [0, \mathrm{C}]$ $(k = 1, 2, \ldots, S)$ is a fractional score which measures the proficiency of e_i for the kth skill. $pro_i^k = 0$ means e_i does not have the kth skill, and $pro_i^k = \mathrm{C}$ shows e_i totally masters the kth skill. According to [12], C is set to be 5 in our experimental study.
$skill_i$	The set of specific skills possessed by e_i. It can be converted from e_i^{skills}, where $skill_i = \{k \mid pro_i^k > 0, k = 1, 2, \ldots, S\}$.
e_i^{maxded}	The maximum dedication of e_i to the project, which means the percentage of a full-time job e_i is able to work. $e_i^{maxded} = 1$ means e_i can dedicate all the normal working hours of a month to the project. Part-time jobs or overtime working are allowed by setting e_i^{maxded} to a value smaller or bigger than 1, respectively. For example, $e_i^{maxded} = 1.2$ indicates e_i is allowed to work up to 120 percent of the normal working time.
$e_i^{norm\_salary}$	The monthly salary of e_i for his/her normal working time.
$e_i^{over\_salary}$	The monthly salary of e_i for his/her overtime working time.
$e_i^{available}(t_l)$	A binary variable which indicates whether e_i is available or not at t_l. $e_i^{available}(t_l) = 1$ means e_i is available at t_l, and $e_i^{available}(t) = 0$ shows e_i is unavailable at t_l.

TABLE 2
Properties of Each Task

name	description
T_j^{skills}	The skill indicator set of task T_j. $T_j^{skills} = \{sk_j^1, sk_j^2, \ldots, sk_j^S\}$, where $sk_j^k = 1$ $(k = 1, 2, \ldots, S)$ indicates the kth skill is required by T_j, and $sk_j^k = 0$ means not.
req_j	The set of specific skills required by T_j. It can be converted from T_j^{skills}, where $req_j = \{k \mid sk_j^k = 1, k = 1, 2, \ldots, S\}$.
$T_j^{est\_tot\_eff}$	The initially estimated effort required to complete task T_j in person-months. The task effort uncertainty of T_j is assumed to follow a normal distribution of $N(\mu_j, \sigma_j)$, where μ_j and σ_j are the mean and standard deviation, respectively. Here, we set $\mu_j = T_j^{est\_tot\_eff}$.
$T_j^{unfinished}(t_l)$	A binary variable indicating whether T_j has finished by t_l. $T_j^{unfinished}(t_l) = 1$ means T_j is unfinished at t_l, and $T_j^{unfinished}(t_l) = 0$ shows T_j has finished by t_l.
TPG	An acyclic directed graph with tasks as nodes and task precedence as edges. TPG must be updated when a task finishes or a new task is added into the project. Here, $G(V(t_l), A(t_l))$ is used to represent the TPG at t_l, where $V(t_l)$ is the vertex set which includes all the arrived and unfinished tasks at t_l, i.e., $V(t_l) = \{T_j \mid T_j^{unfinished}(t_l) = 1, j = 1, 2, \ldots, N_I + N_{new}(t_l)\}$, and $A(t)$ is the arc set which indicates the precedence relations among the tasks in $V(t_l)$.
$T_j^{available}(t_l)$	A binary variable indicating whether T_j is available or not at t_l. $T_j^{available}(t_l) = 1$ shows T_j is available at t_l, while $T_j^{available}(t_l) = 0$ means not. T_j is regarded as available at t_l if and only if the following three conditions are satisfied simultaneously: (1) T_j is unfinished at t_l, i.e., $T_j^{unfinished}(t_l) = 1$; (2) for any skill required by T_j, at least one of the available employees at t_l possesses the skill, i.e., if $k \in req_j$, then $\exists e_i, \text{s.t. } e_i \in e\_ava\_set(t_l) \wedge k \in skill_i$; and (3) all the unfinished tasks preceding T_j in the TPG satisfy the above condition (2).

t_0). Each employee e_i $(i = 1, 2, \ldots, M)$ has some properties (e_i^{skills}, $skill_i$, e_i^{maxded}, $e_i^{norm\_salary}$, $e_i^{over\_salary}$), which are considered to be time-invariant here. During the project, employee e_i may leave, and then come back later. Thus, one time-related variables $e_i^{available}(t_l)$ is also attributed to e_i. Descriptions of an employee's properties are listed in Table 1. $e\_ava\_set(t_l)$ is used to represent the set of all available employees at t_l, i.e., $e\_ava\_set(t_l) = \{e_i | e_i^{available}(t_l) = 1, i = 1, 2, \ldots, M\}$.

3.3 Tasks' Properties

At the initial time t_0, assume there are N_I tasks in the project. As the time progresses, new tasks may be added one by one. At t_l, assume there have been $N_{new}(t_l)$ new tasks arrived. Thus by t_l, a total of $(N_I + N_{new}(t_l))$ tasks have been considered as part of the project. Each task T_j $(j = 1, 2, \ldots, N_I + N_{new}(t_l))$ has some properties (T_j^{skills}, req_j, $T_j^{est\_tot\_eff}$), which are considered to be time-invariant here. At t_l, it is possible that a certain task has finished, or a task cannot be performed temporally because of an employee's leave (one skill required by the task is not possessed by any of the remaining employees). Thus, several time-related properties ($T_j^{unfinished}(t_l)$, TPG, $T_j^{available}(t_l)$) are also attributed to task T_j. Descriptions of a task's properties are listed in Table 2. $T\_ava\_set(t_l)$ is used to represent the set of all available tasks at t_l, i.e., $T\_ava\_set(t_l) = \left\{T_j \middle| T_j^{available}(t_l) = 1, j = 1, 2, \ldots, N_I + N_{new}(t_l).\right\}$.

An example of the TPG update process is shown in Fig. 1. When a task finishes, its corresponding vertex and incident edges are removed from the TPG, e.g., task 1. When a new regular task arrives, it is appended to one or more unfinished tasks, e.g., task 16. If the new task is urgent, its precedence should not be lower than any other unfinished tasks at the time of its arrival. So it may be inserted preceding one or more unfinished tasks, as task 17, or it may be just added as a vertex in the case of not having any precedence relations to other unfinished tasks, as task 18. Note that task preemption is allowed in our MODPSP model. For example, in Fig. 1, since the precedence of the new urgent task 17 is higher than task 3, task 3 should stop processing until task 17 has finished.

An additional property $e_{ij}^{Proficiency}$ of the employee e_i, which indicates the proficiency of e_i for task T_j, is defined according to [12]: $e_{ij}^{Proficiency} = \prod_{k \in req_j} \frac{pro_i^k}{\mathrm{C}}$, and

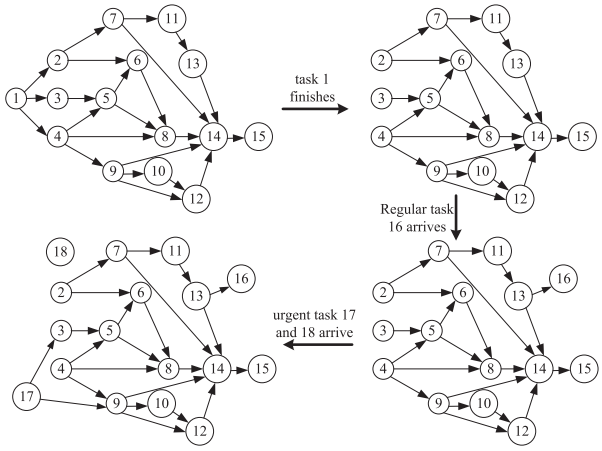


Fig. 1. An example of the update of the TPG.

$e_{ij}^{Proficiency} \in [0,1]$. $e_{ij}^{Proficiency}$ is considered to be time-invariant.

3.4 Solutions to MODPSP

At the scheduling point t_l ($t_l \geq t_0$), a new schedule which determines the dedication matrix $X(t_l) = (x_{ij}(t_l))_{M\times(N_I+N_{new}(t_l))}$ is constructed, where $x_{ij}(t_l)$ denotes the dedication of employee e_i to task T_j scheduled at t_l, and it measures the percentage of a full-time job which e_i spends on T_j. In this paper, $x_{ij}(t_l) \in \{0, e_i^{maxded} \cdot 1/k, \ldots, e_i^{maxded} \cdot k/k\}$, where $k \in \mathbf{N}$ reflects the granularity of the solution, and it is described in Section 5.1 in detail. $x_{ij}(t_l) = 0$ means e_i is not assigned to T_j at t_l. Note that the values of some elements in $X(t_l)$ are determined easily: if $e_i^{available}(t_l) = 0$, then $x_{ij}(t_l) = 0$, for all the $j = 1, 2, \ldots, N_I + N_{new}(t_l)$; if $T_j^{available}(t_l) = 0$, then $x_{ij}(t_l) = 0$, for all the $i = 1, 2, \ldots, M$. Only the values of $x_{ij}(t_l) \in \{x_{ij}(t_l) | e_i^{available}(t_l) = 1 \text{ and } T_j^{available}(t_l) = 1\}$ need to be searched by an optimization method.

3.5 Objectives to be Optimized

At the scheduling point t_l ($t_l > t_0$), considering all the current information gathered from the software project:

- A set of available employees $e\_ava\_set(t_l)$;
- A set of available tasks $T\_ava\_set(t_l)$ with the remaining estimated task efforts. For each task $T_j \in T\_ava\_set(t_l)$, the finished effort from t_0 to t_l is recorded as $T_j^{fin\_eff}(t_l)$. Thus, the remaining estimated effort of T_j at t_l is calculated as $T_j^{est\_rem\_eff}(t_l) = T_j^{est\_tot\_eff} - T_j^{fin\_eff}(t_l)$. If $T_j^{est\_tot\_eff} \leq T_j^{fin\_eff}(t_l)$, but T_j is actually unfinished at t_l, which indicates that the initially estimated effort of T_j is smaller than its actual effort, then the total effort of T_j is re-estimated by sampling a value B from the normal distribution $N(\mu_j, \sigma_j)$ for several times until the condition $B > T_j^{fin\_eff}(t_l)$ is satisfied. Set $T_j^{est\_tot\_eff} = B$, and $T_j^{est\_rem\_eff}(t_l) = T_j^{est\_tot\_eff} - T_j^{fin\_eff}(t_l)$;
- The TPG $G(V(t_l), A(t_l))$ which is updated at t_l,

a new schedule is generated by optimizing the following objectives:

$$\min \mathbf{F}(t_l) = [f_1(t_l), f_2(t_l), f_3(t_l), f_4(t_l)], \tag{1}$$

where $f_1(t_l)$, $f_2(t_l)$, $f_3(t_l)$, and $f_4(t_l)$ are related to the duration, cost, robustness and stability of the project, respectively.

$f_1(t_l)$ represents the maximum elapsed time required for completing the remaining effort of each available task rescheduled at t_l:

$$f_1(t_l) = duration_I = \max_{\{j|T_j \in T\_ava\_set(t_l)\}} \left(T_j^{end}(t_l)\right) - \min_{\{j|T_j \in T\_ava\_set(t_l)\}} \left(T_j^{start}(t_l)\right), \tag{2}$$

where the subscript I denotes the initial scenario, which assumes no task effort variances, and considers the remaining estimated effort $T_j^{est\_rem\_eff}(t_l)$ calculated above as the exact remaining effort of task T_j at t_l. For each available task T_j at t_l, we just consider its remaining effort by t_l, not including its finished effort. Thus, $T_j^{start}(t_l)$ denotes the time (in terms of months) when the remaining effort of T_j starts processing after t_l according to the new generated schedule, but not the starting time of the whole task T_j. Therefore, we have $T_j^{start}(t_l) \geq t_l$; and $T_j^{end}(t_l)$ is the completion time of T_j rescheduled at t_l. According to the TPG and the new schedule (dedication matrix) rescheduled at t_l, we can draw a Gantt chart, from which $T_j^{start}(t_l)$ and $T_j^{end}(t_l)$ of T_j can be obtained.

$f_2(t_l)$ represents the initial cost, which means the total expenses paid to the available employees for their dedications to the available tasks at t_l assuming no task effort variances. Let t' denote any month during which the project is being developed after t_l, and $T\_active\_set(t')$ denote the set of tasks that are active (being developed) at the moment of time t', where an active task is defined as a task that has no preceding unfinished task in the TPG at t'. $f_2(t_l)$ is defined as follows:

$$f_2(t_l) = cost_I = \sum_{t' \geq t_l} \sum_{e_i \in e\_ava\_set(t_l)} e\_cost_i^{t'}. \tag{3}$$

If $\sum_{j \in T\_active\_set(t')} x_{ij}(t_l) \leq 1$, then

$$e\_cost_i^{t'} = e_i^{norm\_salary} \cdot t' \cdot \sum_{j \in T\_active\_set(t')} x_{ij}(t_l), \tag{4}$$

else if $1 < \sum_{j \in T\_active\_set(t')} x_{ij}(t_l) \leq e_i^{maxded}$, then

$$e\_cost_i^{t'} = e_i^{norm\_salary} \cdot t' \cdot 1 + e_i^{over\_salary} \cdot t' \cdot \left(\sum_{j \in T\_active\_set(t')} x_{ij}(t_l) - 1 \right), \tag{5}$$

where $e\_cost_i^{t'}$ means the expense paid to the employee e_i at the moment of time t'. If the total dedications of e_i to all the active tasks ($\sum_{j \in T\_active\_set(t')} x_{ij}(t_l)$) is larger than 1, it indicates that e_i works overtime at t'.

In the following Section 5.3, we will explain how to evaluate the objectives of $duration_I$ and $cost_I$ in detail.

$f_3(t_l)$ represents the robustness performance, which evaluates the sensitivity of a schedule's quality to task effort uncertainties. The smaller the value of $f_3(t_l)$, the better the robustness performance

$$f_3(t_l) = robustness = \sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(\max\left(0, \frac{duration_q(t_l) - duration_I(t_l)}{duration_I(t_l)}\right)\right)^2} + \lambda\sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(\max\left(0, \frac{cost_q(t_l) - cost_I(t_l)}{cost_I(t_l)}\right)\right)^2}, \tag{6}$$

where $duration_I$ and $cost_I$ are the initial duration and cost evaluated from (2) and (3), respectively. Here, the scenario-based method is used. A schedule undergoes a set of task effort scenarios $\{\theta_q | q = 1, 2, \ldots, N\}$, where θ_q is the q^{th} sampled scenario of task efforts, N is the sample size, and we set $N = 30$. $duration_q$ and $cost_q$ are the corresponding efficiency objective values under θ_q. Specifically, at t_l, θ_q is generated as follows: at first, for each task $T_j \in T\_ava\_set(t_l)$, a total effort $T_j^{tot\_eff_q}$ is sampled from the normal distribution $N(\mu_j, \sigma_j)$ at random for multiple times until $T_j^{tot\_eff_q} > T_j^{fin\_eff}(t_l)$ is satisfied, and then set $\theta_q = \{T_j^{rem\_eff_q}(t_l) | T_j^{rem\_eff_q}(t_l) = T_j^{tot\_eff_q} - T_j^{fin\_eff}(t_l), T_j \in T\_ava\_set(t_l)\}$, where $T_j^{rem\_eff_q}(t_l)$ means the q^{th} sampled remaining effort of T_j. Considering that a high efficiency is always addressed in the real-world software project, when defining the robustness objective in (6), we just penalize the duration and cost increases which will cause the efficiency deterioration in the disrupted scenarios, while the variances of duration and cost decreases are truncated by using a "max" function. λ is a weight parameter, which captures the relative importance of the sensitivity of the project cost over the sensitivity of the project duration to task effort uncertainties. λ is set to be 1 in our experiments.

$f_4(t_l)$ denotes the stability, which measures the deviation between the new and original schedules. It is calculated for all the available tasks at t_l ($t_l > t_0$) which are left from the previous schedule created at t_{l-1}. It is defined as the weighted sum of the dedication deviations with the aim of preventing employees from being shuffled around too much

$$f_4(t_l) = stability = \sum_{\{i | e_i \in e\_ava\_set(t_{l-1}) \cap e\_ava\_set(t_l)\}} \sum_{\{j | T_j \in T\_ava\_set(t_{l-1}) \cap T\_ava\_set(t_l)\}} \omega_{ij} | x_{ij}(t_l) - x_{ij}(t_{l-1}) |, \tag{7}$$

where the value of weight ω_{ij} is set as follows:

$$\omega_{ij} = \begin{cases} 2, & \text{if } x_{ij}(t_{l-1}) = 0 \text{ and } x_{ij}(t_l) > 0, \\ 1.5, & \text{if } x_{ij}(t_{l-1}) > 0 \text{ and } x_{ij}(t_l) = 0, \\ 1, & \text{else.} \end{cases} \tag{8}$$

In the first case, a large penalty ($\omega_{ij} = 2$) is given to reschedule an employee to do a new task. If the employee e_i is not assigned to the task T_j at t_{l-1}, but he/she should dedicate to T_j according to the new schedule, then the employee may feel confused. He/she may need additional time to familiarise himself/herself with the newly assigned task, hence the working efficiency may be decreased. In the second case, if an employee was on a task previously, but he/she is not allocated to the task in the new schedule, then a medium penalty ($\omega_{ij} = 1.5$) is given. The employee might have received training about the task and become familiar with the task. Such training would be wasted if the employee does not perform this task any more. In the third case, if an employee continues a task but with a different dedication level, a small penalty ($\omega_{ij} = 1$) is given to the differences between the new and original dedications.

It should be mentioned that at the initial time t_0, only three of the objectives defined above, which are $duration_I$, $cost_I$ and *robustness* (without *stability*), are to be optimized.

3.6 Constraints

Constraints of MODPSP at the scheduling point t_l are listed as follows. Among them, constraints (i)–(ii) are hard constraints, and constraint (iii) is a soft one.

(i) *No overwork constraints.* At the moment of time t' after the scheduling point t_l, the total dedication of an available employee to all the active tasks which are being developed should not exceed his/her maximum dedication to the project, i.e.,

$$\forall\, e_i \in e\_ava\_set(t_l), \forall t' \geq t_l, \text{ s.t. } e\_work_i^{t'} = \sum_{j \in T\_active\_set(t')} x_{ij}(t_l), \text{and } e\_work_i^{t'} \leq e_i^{maxded} \tag{9}$$

For example, if $e_i^{maxded} = 1.2$, then e_i can work overtime, but his/her overtime working should not exceed (120-100 percent) = 20 percent of the normal working time.

(ii) *Task skill constraints.* All the available employees working together for one available task must collectively cover all the skills required by that task, i.e.,

$$\forall\, T_j \in T\_ava\_set(t_l) \text{ s.t. } req_j \subseteq \bigcup\nolimits_{e_i \in e\_ava\_set(t_l)} \{skill_i | x_{ij}(t_l) > 0\} \tag{10}$$

Note that the task skill constraints include the case that each available task at t_l should be performed by at least one available employee.

(iii) *Maximum headcount constraints.* The number of available employees working together for T_j is expected to be no more than an upper limit $T_j^{maxhead}$. Here, $T_j^{maxhead}$ is estimated by the formula in [12]: $T_j^{maxhead} = \max\{1, round(2/3(T_j^{est\_tot\_eff})^{0.672})\}$, which was derived from the COCOMO model [26]. However, if the team size of T_j cannot be reduced to $T_j^{maxhead}$ without violating the task skill constraints, then the maximum headcount constraints can be relaxed, but a penalty should be given to the task effort of T_j which is introduced in Section 5.2.3. At the scheduling point t_l, suppose the team size of T_j is $T_j^{teamsize}(t_l)$, and the minimum number of available employees who should join T_j to satisfy the task skill constraint is $T_j^{\min\_num_{emp}}(t_l)$, then we have:

$$\forall T_j \in T\_ava\_set(t_l), T_j^{teamsize}(t_l) \leq \max\left(T_j^{maxhead}, T_j^{\min\_num_{emp}}(t_l)\right). \tag{11}$$

4 A Proactive-Rescheduling Approach to Solve MODPSP

4.1 Framework of the Proactive-Rescheduling Approach

To handle uncertainties and real-time events occurring during a software project, a proactive-rescheduling approach is proposed for solving the MODPSP. As an illustration for introducing our approach, one real-world instance derived from business software construction projects for a departmental store [8] is taken as an example.

Step i. At the initial time of the project, the software manager identifies several attributes of the project to be developed. These are the tasks, task dependencies, and required efforts. For example, the software manager could identify that there are 12 tasks in total, e.g., performing the UML diagrams, designing the database, designing the web page templates, implementation, testing the software, writing database design documents and a user manual, etc. In fact, if the scheduling process starts after the architecture of the system is designed, it would also be possible to define more fined grained tasks, such as the implementation of each different component of the system. After identifying the tasks, the software manager would identify the dependencies among these tasks by creating a TPG, the skills required by each task, and estimate the effort required for each task. Supporting tools such as COCOMO [26] or machine learning algorithms [27] could be used to help providing effort estimations. Besides the attributes of the project itself, the project manager also identifies employees' properties, such as the skill proficiencies possessed by each employee, the maximum dedication of each employee to the project, the normal and overtime working salaries of each employee. Such information can be obtained based on the experience and knowledge of the software manager on the project. It can also be based on historical information.

Step ii. Provide the information collected in step i as input to the proposed MOEA-based proactive scheduling approach introduced in Section 4.2. The approach will automatically generate schedules to minimize the objectives of project duration (defined by (2)), cost (defined by (3)), and the sensitivity of the schedule to task effort uncertainties (defined by (6)), satisfying the constraints of no overwork, task skills and the maximum headcount (defined by (9)-(11)). The approach assumes that the task effort uncertainties follow a normal distribution. However, a software engineering tool implementing this approach could be easily modified to assume other distributions. The output of the approach is a set of non-dominated solutions which represent schedules with different trade-offs among the three objectives. Each solution is a matrix providing the dedication of each employee to each task.

Step iii. Once the approach generates the non-dominated solutions, the software manager needs to choose one solution to adopt. A tool implementing the approach could display via a GUI some useful information for that, such as the dedication matrix of each solution; its multi-objective values; the maximum, mean and minimum value on each objective among the obtained non-dominated solutions. The software manager could choose the schedule suggested by the automated decision making procedure introduced in Section 4.2.4, or select a schedule manually based on the information provided by our approach and his/her own experience and knowledge about the project. The process of manual decision making that the software manager would need to go through is explained in Section 6.7. After that, the initial project charts, e.g. Gantt charts, can be created using the information of TPG, the estimated task effort and allocation.

Step iv. During the lifetime of the project, some dynamic events may occur, e.g. altering tasks, employee leaves, employee with interrupted involvement, squeeze in budget for some tasks and shift of focus on other tasks. For simplicity, we just consider new task arrivals, employee leaves and employee returns in the current work. Among them, urgent task arrivals, employee leaves and returns are regarded as critical events, while regular task arrivals are considered to be non-critical. To reduce the rescheduling frequency, a critical-event-driven mode is employed. Once a critical event occurs, the software manager triggers the rescheduling procedure provided by our approach. Non-critical events like regular task arrivals are not scheduled until the next critical event occurs. However, if the new regular task needs to start before the next critical event occurs according to the TPG, a heuristic method is used, which assigns a certain number of available employees with higher proficiencies (measured by $e_{ij}^{Proficiency}$) to it, simultaneously satisfying the task skill constraint, and the dedication of each assigned employee to it is generated randomly. This is done automatically, without the need for the software manager to provide manual input.

In the rescheduling procedure which is triggered when a critical event occurs, first, the software manager determines all the available tasks and employees that can be rescheduled in the current environment. The following are provided by the software manager via GUI as the input of the proposed MOEA-based rescheduling approach introduced in Section 4.2: the remaining estimated effort required to finish each available task; the updated TPG reflecting any changes that may have happened to the TPG; other properties of the available tasks and employees, together with the four objectives of duration, cost, robustness and stability defined by (2), (3), (6) and (7), and the three constraints defined by (9)-(11). Such information can also be obtained based on the investigation by and knowledge of the software manager according to the current state of the project. Then the rescheduling approach is triggered, and automatically generates a set of non-dominated solutions, which represent different trade-offs among the four objectives. Next, similar to the steps after proactive scheduling, some useful information is presented via GUI, which the software manager can take as a reference for deciding the final schedule in the new environment. The process of how the software manager would make a decision based on the Pareto front provided by our approach is illustrated in Section 6.7. The new schedule is implemented in the project until the next critical event occurs, at which time the above rescheduling

TABLE 3
Relationship between Four Kinds of Dynamic Features Considered in Our Approach and Five Reasons for Late Software Delivery Noted by Pressman

Five reasons [14]	Dynamic events	Treatment
Changing customer requirements that are not reflected in schedule changes.	Task effort uncertainties, new task arrivals.	Proactiveness/ rescheduling.
An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.	Task effort uncertainties.	Proactiveness.
Predictable and/or unpredictable risks that were not considered when the project commenced.	New task arrivals, employee leaves, employee returns.	Rescheduling.
Technical difficulties that could not have been foreseen in advance.	Task effort uncertainties.	Proactiveness.
Human difficulties that could not have been foreseen in advance.	Employee leaves, employee returns.	Rescheduling.

procedure is triggered again. In short, the MODPSP is a dynamic process formed by a sequence of multi-objective PSPs with different sets of available employees and tasks to be scheduled. This process continues until the whole project has been completed.

As indicated in Section 1, five of the eight reasons given by Pressman [14] for late software delivery are related to uncertainties, risks and unpredictable events appearing during the project execution. In our current work, four kinds of risks or dynamic events including task effort uncertainties, new task arrivals, employee leaves, and employee returns are considered. Each of the above four cases can be linked to one of the five reasons for late software delivery noted by Pressman. The relationship between them and the strategies used by our proactive-rescheduling approach to address these issues are shown in Table 3.

4.2 An MOEA-Based Rescheduling Method for MODPSP

The goal of multi-objective optimization is to find a representative set of Pareto non-dominated solutions. One solution is said to Pareto dominate another if the first is not worse than the second in all objectives, and there is at least one objective where it is better. A solution is called Pareto non-dominated if none of the objectives can be improved without sacrificing some of the other objective values. The set of Pareto non-dominated solutions in the objective space is called the Pareto front, which can provide trade-offs among multiple objectives.

ε-MOEA is an ε-domination based MOEA [28], where ε-domination is a generalization of the domination relation introduced in [29]. ε-MOEA employs efficient parent and archive update strategies, and can produce good convergence and diversity with a small computational effort, especially when dealing with many objectives (3 or more) [28]. MODPSP is a dynamic problem with four objectives. In order to solve it in an efficient way, an ε-MOEA-based rescheduling method, dε-MOEA, is proposed in this paper.

Step 1: Initialization. Construct the initial population $P(t_l)$ by some heuristic strategies (they are described in Section 4.2.3) according to the updated project state at t_l. Sample a set of task effort scenarios θ_q at random, $q = 1, 2, \ldots, N$. Then multi-objective evaluations are performed, and all the Pareto non-dominated solutions are determined to form the archive population $Arc(t_l)$. Set the counter of objective evaluation numbers $ct = population\_size$.

Step 2: Population selection. One individual sp is chosen from the population $P(t_l)$ using a *pop_selection* procedure.

Step 3: Archive selection. One solution e is chosen from the archive $Arc(t_l)$ using the *archive_selection* procedure.

Step 4: Variation. Two offspring sc_1 and sc_2 are generated from sp and e by the variation operators.

Step 5: Decoding and objective evaluation. Sample a set of task effort scenarios θ_q at random, $q = 1, 2, \ldots, N$. Evaluate the multiple objective values of offspring sc_1 and sc_2.

Step 6: Update of the population. Offspring individuals sc_1 and sc_2 are included in $P(t_l)$ using a *pop_acceptance* procedure.

Step 7: Update of the archive. Individuals sc_1 and sc_2 are included in $Arc(t_l)$ using an *archive_acceptance* procedure.

Step 8: Termination. If the termination criterion is not satisfied, set $ct = ct + 2$ and go to Step 2. Otherwise, determine all the Pareto non-dominated solutions from $Arc(t_l)$, record it as $Arc'(t_l)$, output $Arc'(t_l)$, and select one solution from $Arc'(t_l)$ as the implementation schedule based on a decision making procedure.

Fig. 2. Procedure of dε-MOEA at the scheduling point t_l ($t_l > t_0$).

4.2.1 The Procedure of dε-MOEA Applied to MODPSP

At each scheduling point t_l ($t_l > t_0$) in MODPSP, the procedure of dε-MOEA is presented in Fig. 2.

For Step 1, update of the project state and heuristic constructions of the initial population are described in Sections 4.2.2 and 4.2.3, respectively. The tournament selection method is used for the *pop_selection* procedure in Step 2. Two individuals are picked up uniformly at random from the population, and check the domination of each other. If one dominates the other, the former will be chosen. Otherwise, one of them is selected at random. In Step 3, an individual is selected uniformly at random from the archive. In Step 4, the variation operators are introduced in Section 5.1. In Step 5, the sampled task efforts change from one iteration to another, which increases the probability of generating robust solutions undergoing a large number of scenarios. The *pop_acceptance* and *archive_acceptance* procedures in Steps 6 and 7 are the same as in [28]. The termination criterion is that the counter ct achieves a predefined maximum number of objective evaluations. The decision making procedure is described in Section 4.2.4. For each candidate solution, the constraint handling methods and objective evaluation procedure are presented in Sections 5.2 and 5.3, respectively.

It should be mentioned that at the initial time t_0 of the project, the proactive scheduling is also based on the dε-

MOEA procedure shown in Fig. 2. The differences are the random population initialization is used in Step 1 instead of the heuristic population initialization, and when evaluating an individual, only three objectives (without *stability*) are considered.

4.2.2 Update of the Project State

At each scheduling point t_l ($t_l > t_0$), the project state should be updated first.

(i) The finished effort of each task from t_0 to t_l should be calculated. If a task has been completed by t_l, its corresponding vertex and incident edges are removed from the TPG.

(ii) Information about the new tasks arriving since the previous scheduling point t_{l-1} must be gathered. The new tasks and their task precedence are added into the TPG.

(iii) For each task, whether it is available or not at t_l is determined by checking the three conditions introduced in Table 2.

As a result of the above three steps, all the current available employees, available tasks, and the updated TPG can be used for rescheduling at t_l.

4.2.3 Heuristic Population Initialization in Rescheduling

With the aim of utilizing the dynamic features of MODPSP and accelerating the convergence speed of the algorithm, several heuristic strategies are incorporated in constructing the initial population of dε-MOEA.

(1) *Exploitation of the dynamic event characteristics.* Inspired by the schedule repair often used in production scheduling, which refers to local adjustments to the original schedule and has the ability of preserving the system stability well [15], three schedule repair strategies are specifically designed for MODPSP to exploit the dynamic event features. Firstly, in the case of employee leaves, all the unaffected tasks remain unchanged both for their employees and dedications. For each affected task to which the leaving employee was assigned, the condition of whether the remaining employees in the task team can satisfy the task skill constraint is checked. If yes, their dedications to the task are kept unchanged. Otherwise, other available employees with relatively higher proficiencies are found to join the task team to satisfy the skill requirement. Secondly, in the case that an employee returns, for each task left from the previous schedule, if its team size is less than the maximum headcount, and the returning employee has one of the task skills, then he/she is assigned to the task to speed up the task progress. Otherwise, the previously scheduled employees and dedications remain unchanged. For each new arriving regular task, or each previously unavailable task that becomes available again due to the employee return, the dedications of the available employees to it are generated at random. Thirdly, in the case of new urgent task arrivals, the employees and their dedications assigned to each task left from the previous schedule are kept unchanged, while the dedications to the new tasks are generated at random. In the above cases, if overwork of any available employee appears, the normalization method explained in 5.2.1 is applied. The result of the schedule repair is called the schedule repair solution.

(2) *Exploitation of the history information.* At each scheduling point, information left from the previous schedule is regarded as the history information which can be utilized. The dedication allocations of the available employees to the available tasks in the old schedule are called the history solution.

(3) *Incorporation of random individuals.* In order to introduce diversity, some random individuals are created in the initial population. The dedication of each available employee to each available task is generated uniformly at random from the set $\{0, e_i^{maxded}(t_l) \cdot 1/k, \ldots e_i^{maxded}(t_l) \cdot k/k\}$.

In this paper, 20 percent of the initial population are formed with the history solution and its variants by mutation, 30 percent with the schedule repair solution and its variants, and 50 percent with the random individuals.

4.2.4 Decision Making

In practice, at each scheduling point, once a set of non-dominated solutions are found by dε-MOEA, they are provided to the software manager for selection, and then the selected schedule is implemented in the project. However, in our experiments, it is not practical to have a person for taking decisions. Thus, an automatic decision making method proposed in our previous work [30] is adopted, and the procedure is briefly given as follows.

Step i. Construction of the pairwise comparison matrix. Our MODPSP uses $N\_o = 4$ objectives to be optimized. The pairwise comparison questions of "How important is the objective f_i relative to f_j ?" ($i, j = 1, 2, \ldots, N\_o$, $j > i$) are answered by the software manager a priori. So there are $N\_o \cdot (N\_o - 1)/2 = 4(4-1)/2 = 6$ comparisons in total in our case. Then the pairwise comparison matrix $C_1 = (c_{ij})_{N\_o \times N\_o}$ can be constructed by the nine-point scale in Analytic Hierarchy Process (AHP) [31], which describes the degree of the preference for one objective versus another.

Step ii. Estimation of the weight vector $w = (w_i)_{N\_o \times 1}$ for multiple objectives. The logarithmic least squares method [32] is adopted. The geometric mean of each row in the matrix C_1 is calculated, which is then normalized by dividing it by the sum of them.

Step iii. Normalization of the objective values. Each objective is normalized as

$$n\_f_i(x) = \left(f_i^{\max} - f_i(x)\right) / \left(f_i^{\max} - f_i^{\min}\right), \quad i = 1, 2, \ldots, N\_o, \quad (12)$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum objective values among all the non-dominated solutions obtained at the current scheduling point.

Step iv. Calculation of the utility value. The weighted geometric mean of the multiple objective values is used to find the utility value for each non-dominated solution:

$$U(x) = \prod_{i=1}^{N\_o} n\_f_i(x)^{w_i / \sum_i^{N\_o} w_i}. \quad (13)$$

Step v. Choose the solution with the maximum utility value as the final schedule.

Note that the pairwise comparison matrix and the weight vector in Steps i and ii are determined beforehand and kept unchanged during the dynamic process. Only Steps iii, iv, and v are performed at each scheduling point during the project execution.

Here, we give an example of the above decision making method. Before the beginning of the project, assume that the software manager considers that the objectives $duration_I$ and $cost_I$ are of the equal importance; *robustness* and *stability* are of the equal importance; and the intensity scale of the importance of $duration_I$ (or $cost_I$) over *robustness* (or *stability*) is set as the intermediate value between equal importance and weak importance. Thus the pairwise comparison matrix for the four objectives is constructed according to the nine-point scale in AHP:

$$\mathrm{C}_1 = (c_{ij})_{4\times4} = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1/2 & 1/2 & 1 & 1 \\ 1/2 & 1/2 & 1 & 1 \end{bmatrix},$$

and $\mathrm{w} = (w_i)_{4\times1} = [0.3333 0.3333 0.1667 0.1667]^T$ can be obtained according to the above Step ii. At each scheduling point during the project execution, after normalizing each objective according to (12), the utility value of each non-dominated solution can be calculated based on (13). Then, the non-dominated solution with the highest utility value is chosen.

There have been AHP-related decision making methods in the existing work. Javanbarg et al. [33] proposed a fuzzy AHP decision making model to deal with the imprecise judgments of decision makers, and then a fuzzy prioritization method was applied to derive exact priorities from consistent and inconsistent fuzzy comparison matrices. Kim and Langari [34] gave an adaptive AHP for decision making in the dynamically changing traffic environment, which could provide an optimal relative importance matrix under different traffic situations and driving modes. Bernardon et al. [35] presented a multicriteria decision-making process for solving the remote-controlled switch allocation problem based on AHP. In [33], the weight of each objective was set as the algebraic mean of each row in the normalized pairwise comparison matrix C_1, and in [34], the weight was set as the element in the eigenvector associated with the maximum eigenvalue of C_1. Both [33] and [34] used the weighted algebraic mean to evaluate the utility of each alternative. However, when estimating the weight vector $\mathrm{w} = (w_i)_{N\_o\times1}$, it is expected that the entry $W_{ij} = w_i/w_j$ in the matrix $\mathrm{W} = (W_{ij})_{N\_o\times N\_o}$ will provide the best fit to the judgement c_{ij} in C_1 [31]. Thus, in our work, the logarithmic least squares method is used to calculate the weight vector based on the minimization of the distance between C_1 and W. Meanwhile, the weighted geometric mean, which is considered as the optimal method to find the utility value for the alternative [36], is employed.

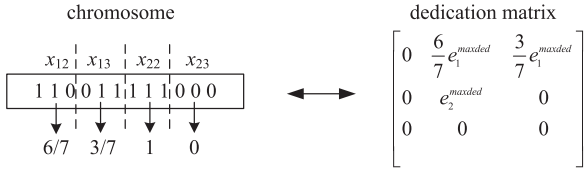


Fig. 3. An example of the representation of a chromosome and its decoded dedication matrix.

5. Details of Our Implementation

5.1 Representations and Variation Operators

In MODPSP, the solution at each scheduling point t_l is a dedication matrix $\mathrm{X}(t_l) = \left(x_{ij}(t_l)\right)_{M\times(N_I+N_{new}(t_l))}$, where $x_{ij}(t_l) \in [0, e_i^{maxded}]$. We employ binary string chromosomes to encode solutions in dε-MOEA. nb bits are used to represent an $x_{ij}(t_l)$, so that $x_{ij}(t_l) \in \{0, e_i^{maxded}\cdot 1/k, \ldots, e_i^{maxded}\cdot k/k\}$, $k = 2^{nb} - 1$. As mentioned in Section 3.4, in $\mathrm{X}(t_l)$, only the values of $x_{ij}(t_l) \in \{x_{ij}(t_l) | e_i^{available}(t_l) = 1 \text{ and } T_j^{available}(t_l) = 1\}$ have to be searched, while other elements should be 0. In order to improve the efficiency of dε-MOEA, only such$x_{ij}(t_l)$are encoded in the chromosome, which has a length of $|e\_ava\_set(t_l)| \cdot |T\_ava\_set(t_l)| \cdot nb$ bits ($|\cdot|$ means cardinality of a set). The chromosome should be decoded into a dedication matrix for the convenience of objective evaluation. Fig. 3 gives an example of the representation of a binary chromosome and its decoded dedication matrix, where there are two available employees e_1, e_2, two available tasks T_2, T_3, one leaving employee e_3, one finished task T_1, and $nb = 3$.

In dε-MOEA, the 2-D single point crossover operator [3], which is designed for matrices, and the bit-flip mutation are employed as variation operators.

5.2 Constraint Handling

5.2.1 Handling the No Overwork Constraints

In [3], overwork is handled by penalizing the fitness value of a schedule. As shown in their experimental results, the no overwork constraints are difficult to be satisfied by this method, especially when the number of tasks or employees is increased, or the employees' skills are decreased, or the project demands more skills. A modification to the dedication normalization method proposed in [2] is employed here.

At time t', if the no overwork constraint for the employee e_i is violated, i.e., $e\_work_i^{t'} > e_i^{maxded}$, then his/her dedication $x_{ij}(t_l)$ to each active task T_j, which is being performed at t', is divided by $e\_work_i^{t'}/$. If $e\_work_i^{t'} \le e_i^{maxded}$, then the dedication is not normalized. The normalized value of the dedication $x_{ij}(t_l)$ is denoted as $d_{ij}(t_l)$, and we have $d_{ij}(t_l) = x_{ij}(t_l)/\max(1, e\_work_i^{t'}/e_i^{maxded})$. The normalized dedications $d_{ij}(t_l)$ are the ones that employees will use at any moment after t_l in order to avoid overwork. This

method allows an employee to divide his/her dedications to several tasks, and it is guaranteed that the no overwork constraints can always be satisfied by such an adjustment.

5.2.2 Handling the Task Skill Constraints

In order to incorporate the proficiency of each employee for different tasks when evaluating a schedule and handling the task skill constraints, according to [10] and [12], the adjusted total dedication $A\_Td_j(t_l)$ for task T_j can be calculated as follows:

First, the total dedication $Td_j(t_l)$of all the available employees for T_j is

$$Td_j(t_l) = \sum_{e_i \in e\_ava\_set(t_l)} d_{ij}(t_l). \tag{14}$$

Second, the total fitness $F_j(t_l)$ of all the available employees for the task T_j is calculated:

$$F_j(t_l) = \left(\sum_{e_i \in e\_ava\_set(t_l)} e_{ij}^{Proficiency} \cdot d_{ij}(t_l) \right) / Td_j(t_l), \tag{15}$$

where $F_j(t_l)$ is a fraction of the total dedication spent by employees to the task T_j. The explanation for this is as follows. Even though employees have a dedication of $d_{ij}(t_l)$ for the task T_j, if their proficiency on the skills needed for the task are low, T_j will take longer to finish, as if the employees' dedications were lower than $d_{ij}(t_l)$. (15) reduces the dedications of employees to tasks based on their proficiency.

Third, $F_j(t_l)$ is converted to a cost drive value $V_j(t_l)$:

$$V_j(t_l) = \max\big(1, 8 - round\big(F_j(t_l) * 7 + 0.5\big)\big), \tag{16}$$

where the value of $V_j(t_l)$ ranges from 1 to 7. $V_j(t_l) = 1$ indicates the assigned employees are the most suitable for task T_j, and vice versa. This conversion was proposed by [12].

Fourth, the adjusted total dedication $A\_Td_j(t_l)$, which takes into account the proficiency of the employees, can be obtained:

$$A\_Td_j(t_l) = Td_j(t_l)/V_j(t_l), \tag{17}$$

where $A\_Td_j(t_l)$ is in person.

Assume $T_j^{rem\_eff}(t_l)$ is the remaining effort of task T_j at t_l, then the time required to finish T_j is

$$T_j^{rem\_eff}(t_l)/A\_Td_j(t_l) = T_j^{rem\_eff}(t_l)/\big(Td_j(t_l)/V_j(t_l)\big). \tag{18}$$

At the scheduling point t_l, if a candidate schedule is infeasible because certain task skills are not covered by the allocated employees, then very high penalty values are assigned to the objectives, as suggested in [2]. Suppose $reqsk$ is the number of missing skills in an infeasible schedule. Each objective is penalized as follows:

$$\begin{aligned} f_1(t_l) &= duration_I \\ &= reqsk \cdot 2 \cdot \sum_{T_j \in T\_ava\_set(t_l)} T_j^{est\_rem\_eff}(t_l) \Big/ \\ &\left(\min_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}/k/ \max_{T_j \in T\_ava\_set(t_l)} V_j \right) \\ &= reqsk \cdot 2 \cdot \sum_{T_j \in T\_ava\_set(t_l)} T_j^{est\_rem\_eff}(t_l) \Big/ \\ &\left(\min_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}/k/7) \right) \\ &= reqsk \cdot 14k \cdot \sum_{T_j \in T\_ava\_set(t_l)} T_j^{est\_rem\_eff}(t_l) \Big/ \\ &\min_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}, \end{aligned} \tag{19}$$

$$\begin{aligned} f_2(t_l) &= cost_I \\ &= reqsk \cdot 2 \cdot \sum_{e_i \in e\_ava\_set(t_l)} \sum_{T_j \in T\_ava\_set(t_l)} e_i^{over\_salary} . T_j^{est\_rem\_eff}(t_l) \cdot 7 \\ &= reqsk \cdot 14 \cdot \sum_{e_i \in e\_ava\_set(t_l)} \sum_{T_j \in T\_ava\_set(t_l)} e_i^{over\_salary} . T_j^{est\_rem\_eff}(t_l), \end{aligned} \tag{20}$$

$$f_3(t_l) = robustness = reqsk \cdot 2 \cdot C_{rob}, \tag{21}$$

$$\begin{aligned} f_4(t_l) &= stability \\ &= reqsk \cdot 2 \cdot |e\_ava\_set(t_l)| \cdot |T\_ava\_set(t_l)| \cdot \max_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}, \end{aligned} \tag{22}$$

where C_{rob} is a constant, and we set $C_{rob} = 100$ here.

All the four penalized values are higher than the corresponding objective values of any feasible schedule, since, at t_l:

- The duration is always at most $7k \cdot \sum_{T_j \in T\_ava\_set(t_l)} T_j^{est\_rem\_eff}(t_l) / \min_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}$. The explanation for this is as follows. In the worst case, tasks are processed one by one. The total dedication for each task is the minimum value $\min_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}/k$, and the cost driver value of each task takes the maximum value 7.
- The cost is always at most $\sum_{e_i \in e\_ava\_set(t_l)} \sum_{T_j \in T\_ava\_set(t_l)} e_i^{over\_salary} . T_j^{est\_rem\_eff}(t_l) \cdot 7$. The explanation for this is as follows. In the worst case, all the available employees have to dedicate to all tasks with his/her overwork salary $e_i^{over\_salary}$. Moreover, the total dedication of each employee to each task equals to the total effort required for this task $T_j^{est\_rem\_eff}(t_l) \cdot 7$, where 7 is the maximum possible cost driver value of each task. This is the total dedication as if each employee was the only employee working for the task, i.e., the maximum possible total dedication of the employee for the task.
- The *stability* value is always at most $|e\_ava\_set(t_l)| \cdot |T\_ava\_set(t_l)| \cdot \max_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}$. In the worst case, dedication deviations of all the available

employees to all the available tasks are $\max_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}$.

- The $robustness$value was always much smaller than the constant C_{rob} from our experimental observations.
- Moreover, the penalty values are proportional to the value of $reqsk$, which means the penalty will decrease if the number of missing skills decreases. This penalized objective vector gives a strong gradient for search algorithms towards feasible regions.

5.2.3 Handling the Maximum Headcount Constraints

In order to improve the efficiency of our algorithm, two heuristic operators are performed for a candidate schedule before the objective evaluation. The first one is to set the dedication of an employee for a task to 0 if he/she has none of the skills required by the task, i.e., if $e_{ij}^{Proficiency} = 0$, then set $x_{ij}(t_l) = 0$.

The second one is to check whether the team size of each available task $T_j \in T\_ava\_set(t_l)$ is larger than its maximum headcount $T_j^{maxhead}$. If $T_j^{maxhead}$ is exceeded, then the following procedure is performed: 1) sort the proficiency $e_{ij}^{Proficiency}$ of all the employees in the team of T_j; 2) start from the employee with the lowest proficiency and have a check. If removing him/her does not violate the task skill constraints, then he/she can be removed (set the corresponding $x_{ij}(t_l) = 0$), otherwise, he/she is kept in the team; 3) move to the next employee in the sorting list and do the same operation as in 2) for him/her. This procedure continues until the team size of T_j is within the limit or all the employees in the team have been checked. If the team size cannot be reduced to $T_j^{maxhead}$ without violating the task skill constraints, then it can be larger than $T_j^{maxhead}$, but a penalty is given to the effort of T_j. As indicated in [37], the communication overhead must be added to the amount of work to be done. If each part of the task has to be separately coordinated with each other part, then the effort requires $T_j^{teamsize}\left(T_j^{teamsize}-1\right)/2$ times as much pairwise intercommunication as that of having only two employees in the task, where $T_j^{teamsize}$ denotes the number of employees assigned to T_j. Thus, if $T_j^{teamsize} > T_j^{maxhead}$, then we give a penalty to the effort of T_j as follows:

$$T_j^{eff} = T_j^{eff} \cdot \left(1 + \frac{T_j^{teamsize} \cdot (T_j^{teamsize} - 1)/2}{Z}\right), \qquad (23)$$

where T_j^{eff} is the effort of T_j without considering the overhead, and Z is a parameter. We have performed some preliminary experiments and found that when Z = 5, the relationship of the time to finish the task versus the number of employees has a similar behavior to the curve of Fig. 2.4 shown in [37].

In [10] and [12], the maximum headcount constraints were also considered. However, neither of them presented any method to handle such constraints, which might produce infeasible solutions and introduce communication overheads. Here, the approach of penalizing the task effort given in (23) fills the gap in the literature.

5.3 Objective Evaluations

The pseudo code of the objective evaluation procedure at the scheduling point t_l is given in Fig. 4. At first, the procedure tests whether the dedication matrix $\mathrm{X}(t_l)$ is feasible in that the task skill constraint of every available task is satisfied (lines 1-4). If there is no missing skills ($reqsk = 0$), two heuristic operators introduced in Section 5.2.3 are performed, and the modified dedication matrix $\mathrm{X}'(t_l)$ is obtained (lines 5-6). If the team size of T_j is still bigger than $T_j^{maxhead}$ after the heuristic operators, then give a penalty to the effort of T_j (lines 7-12). For $\mathrm{X}'(t_l)$, two efficiency objectives of $duration_I$ and $cost_I$ are evaluated by calling the function Evaluate_duration_cost (line 16), the procedure of which is given in Fig. 5. If the task skill constraints are violated ($reqsk > 0$), output the penalized objective vector (lines 17-19) and stop the procedure. Otherwise, the robustness and stability values of $\mathrm{X}'(t_l)$ are calculated (lines 20-36). Note that the modified dedication matrix $\mathrm{X}'(t_l)$ is also output by the procedure, which will replace $\mathrm{X}(t_l)$ in the succeeding optimization.

The procedure of evaluating duration and cost shown in Fig. 5 is a modification to Algorithm 1 in [2], which provides a schedule-driven estimation [38] of duration and cost. Algorithm 1 considered all the tasks and employees in the static PSP, while our procedure here is for computing the elapsed time and cost of processing the available tasks by the available employees at a specific scheduling point. Moreover, the skill proficiencies and overtime salaries are taken into account in our work. If the task skill constraints are satisfied, the procedure iteratively constructs the schedule (Lines 5-34). Line 6 checks which tasks can be active at the current moment of time according to the TPG. The dedication is normalized for employees whose total dedication to all the active tasks exceeds the upper limit (line 12). Next, the total dedication of employees for a task is calculated (Line 14), and the total fitness for the task is evaluated and converted to a cost drive value by (15) and (16) (lines 15-16). Line 18 determines the earliest moment of time t' at which a task finishes. The finished task and its incident edges are removed from the TPG (line 31), thus new tasks are allowed to become active in the next iteration based on the TPG. Duration and cost are accumulated along all the iterations (lines 19–27). Line 29 computes the remaining effort of each active task, which will be used in the next iteration if the task has not finished yet.

6 Experimental Studies

Considering the uncertainties and dynamic events that often occur in dynamic environments of a software project, it is desirable to provide the software manager with insight into whether robustness and stability should be taken into account together with project duration and cost, and which rescheduling method to choose for solving MODPSP. This insight should be supported by evidences illustrating the influence of robustness and stability on the project duration and cost and also on the

Procedure 1 Evaluatė_objective

Input: $T\_ava\_set(t_l)$, $e\_ava\_set(t_l)$, $G(V(t_l),A(t_l))$, $T_j^{est\_rem\_eff}(t_l)$, $T_j^{rem\_eff_q}(t_l)$, $\mathrm{X}(t_l)$, $\mathrm{X}(t_{l-1})$, $T\_ava\_set(t_{l-1})$, $e\_ava\_set(t_{l-1})$

// $G(V(t_l),A(t_l))$ is the TPG at t_l , and $\mathrm{X}(t_l)$ is the dedication matrix at t_l .

Output: ($duration_l(t_l)$, $cost_l(t_l)$, $robustness(t_l)$, $stability(t_l)$, $\mathrm{X}'(t_l)$)

1: Let $reqsk := 0$.

2: **for** all tasks T_j in $T\_ava\_set(t_l)$ **do**

3: $reqsk := reqsk + \left| req_j \setminus \bigcup_{e_i \in e\_ava\_set(t_l)} \{skill_t \mid x_{ij} > 0\} \right|$. // $reqsk$ is the total number of missing skills

4: **end for**

5: **if** $reqsk = 0$ **then**

6: Perform the two heuristic operators introduced in Section 5.2.3, and obtain the modified dedication matrix $\mathrm{X}'(t_l)$.

7: **for** all tasks T_j in $T\_ava\_set(t_l)$ **do**

8: **if** $teamsize_j$ of T_j after performing heuristic operators exceeds $T_j^{maxhead}$, **then**

9: $$T_j^{est\_rem\_eff}(t_l) := T_j^{est\_rem\_eff}(t_l) \cdot \left(1 + \frac{teamsize_j \cdot (teamsize_j - 1)/2}{Z}\right);$$

10: $$T_j^{rem\_eff_q}(t_l) := T_j^{rem\_eff_q}(t_l) \cdot \left(1 + \frac{teamsize_j \cdot (teamsize_j - 1)/2}{Z}\right);$$

11: **end if**

12: **end for**

13: **else**

14: Let $\mathrm{X}'(t_l) := \mathrm{X}(t_l)$

15: **end if**

16: Call the function:

[$duration_l(t_l)$, $cost_l(t_l)$]:=Evaluate_duration_cost($T\_ava\_set(t_l)$, $e\_ava\_set(t_l)$, $G(V(t_l),A(t_l))$, $T_j^{est\_rem\_eff}(t_l)$, $\mathrm{X}'(t_l)$, $reqsk$).

// $T_j^{est\_rem\_eff}(t_l)$ represents the initial scenario.

17: **if** $reqsk > 0$ **then**

18: Output ($duration_l(t_l)$, $cost_l(t_l)$, $reqsk \cdot 2 \cdot C_{rob}$, $reqsk \cdot 2 \cdot |e\_ava\_set(t_l)| \cdot |T\_ava\_set(t_l)| \cdot \max_{e_i \in e\_ava\_set(t_l)} e_i^{maxded}$, $\mathrm{X}'(t_l)$); exit.

19: **end if**

20: **for** all the sampled task effort scenarios θ_q in $\{\theta_q \mid q = 1,2,\ldots,N\}$ **do** // N is the sample size

21: Call the function:

[$duration_q(t_l)$, $cost_q(t_l)$]:=Evaluate_duration_cost($T\_ava\_set(t_l)$, $e\_ava\_set(t_l)$, $G(V(t_l),A(t_l))$, $T_j^{rem\_eff_q}(t_l)$, $\mathrm{X}'(t_l)$, $reqsk$).

// $T_j^{rem\_eff_q}(t_l)$ represents the q^{th} sampled scenario.

22: **end for**

23: Let $$robustness(t_l) := \sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(\max\left(0, \frac{duration_q(t_l) - duration_l(t_l)}{duration_l(t_l)}\right)\right)^2} + \lambda\sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(\max\left(0, \frac{cost_q(t_l) - cost_l(t_l)}{cost_l(t_l)}\right)\right)^2}.$$ // $\lambda = 1$

24: Let $stability(t_l) := 0$.

25: **for** all employees e_i in $e\_ava\_set(t_{l-1}) \cap e\_ava\_set(t_l)$ **do**

26: **for** all tasks T_j in $T\_ava\_set(t_{l-1}) \cap T\_ava\_set(t_l)$ **do**

27: **if** $x_{ij}(t_{l-1}) = 0$ and $x'_{ij}(t_l) > 0$ **then**

28: Let $\omega_{ij} := 2$;

29: **else if** $x_{ij}(t_{l-1}) > 0$ and $x'_{ij}(t_l) = 0$ **then**

30: Let $\omega_{ij} := 1.5$;

31: **else**

32: Let $\omega_{ij} := 1$.

33: **end if**

34: Let $stability(t_l) := stability(t_l) + \omega_{ij} \cdot |x'_{ij}(t_l) - x_{ij}(t_{l-1})|$

35: **end for**

36: **end for**

37: Output ($duration_l(t_l)$, $cost_l(t_l)$, $robustness(t_l)$, $stability(t_l)$, $\mathbf{X}'(t_l)$); exit.

Fig. 4. Pseudo code of the objective evaluation procedure at the scheduling point t_l.

performance of the algorithm. The evidence should also demonstrate which rescheduling algorithm is likely to behave better according to the evaluation criteria that may affect the software manager's decision. With this aim, this section presents a comprehensive study of the influence of the robustness objective on proactive scheduling, compares our rescheduling method, dε-MOEA, with the heuristic dynamic scheduling based on the whole project duration and cost, and also compares five MOEA-based rescheduling methods based on the

Procedure 2 Evaluate_duration_cost

Input: T_ava_set, e_ava_set, $G(V,A)$, $T_j^{rem\_eff}$, X, $reqsk$ // $T_j^{rem\_eff}$ is the remaining effort of task T_j

Output: ($duration$, $cost$)

1: **if** $reqsk > 0$ **then** // $reqsk$ is the total number of missing skills

2: Output ($reqsk \cdot 14k \cdot \sum_{T_j \in T\_ava\_set(t_1)} T_j^{est\_rem\_eff}(t_1) \Big/ \min_{e_i \in e\_ava\_set(t_1)} e_i^{maxded}$, $reqsk \cdot 14 \cdot \sum_{e_i \in e\_ava\_set} \sum_{T_j \in T\_ava\_set} e_i^{over\_salary} \cdot T_j^{rem\_eff}$); exit.

3: **end if**

4: Let $duration := 0$, $cost := 0$.

5: **while** $G(V,A) \neq \varnothing$, **do**

6: Let V' be the set of all the tasks in T_ava_set without incoming edges in $G(V,A)$.

7: **if** $V' = \varnothing$ **then**

8: Output "Problem instance not solvable!"; exit.

9: **end if**

10: **for** all tasks T_j in V' **do**

11: **for** all employees e_i in e_ava_set **do**

12: Let $d_{ij} := x_{ij} \Big/ \max\left(1, \sum_{T_m \in V'} x_{im} \Big/ e_i^{maxded}\right)$. //*normalization*

13: **end for**

14: Compute the total dedication for T_j: $Td_j := \sum_{e_i \in e\_ava\_set} d_{ij}$;

15: Compute the total fitness for T_j: $F_j := \left(\sum_{e_i \in e\_ava\_set} e_{ij}^{Proficiency} \cdot d_{ij}\right) \Big/ Td_j$;

16: Convert F_j to a cost drive value $V_j := \max\left(1, 8 - round\left(F_j * 7 + 0.5\right)\right)$; // $V_j = 1$ *means the employees are the most suitable for* T_j.

17: **end for**

18: Let $t' := \min_{T_j \in V'}\left(T_j^{rem\_eff} \Big/ \left(Td_j / V_j\right)\right)$.

19: Let $duration := duration + t'$.

20: **for** all employees e_i in e_ava_set **do**

21: Compute the total dedication of e_i: $Ed_i := \sum_{T_j \in V'} d_{ij}$;

22: **if** $Ed_i \leq 1$ **then** //within the normal working time

23: Let $cost := cost + t' \cdot e_i^{norm\_salary} \cdot Ed_i$;

24: **else** //overtime working time

25: Let $cost := cost + t' \cdot e_i^{norm\_salary} \cdot 1 + t' \cdot e_i^{over\_salary} \cdot (Ed_i - 1)$.

26: **end if**

27: **end for**

28: **for** all tasks T_j in V' **do**

29: Let $T_j^{rem\_eff} := T_j^{rem\_eff} - t' \cdot \left(Td_j / V_j\right)$.

30: **if** $T_j^{rem\_eff} = 0$ **then**

31: Mark T_j as finished, and remove it and its incident edges from $G(V,A)$.

32: **end if**

33: **end for**

34: **end while**

35: Output (*duration*, *cost*); exit.

Fig. 5. Pseudo code of evaluating duration and cost.

convergence, distribution and spread performances that are usually considered in multi-objective optimization.

6.1 MODPSP Instances

In our experiments, both the instances derived from Alba and Chicano's benchmarks [3], and those derived from real-world projects were used.

Since there are no standard benchmarks for the MODPSP, 18 dynamic instances are generated based on the 18 static PSP instances of benchmark 4 in [3]. The reason for selecting these 18 instances is that they include the variants of three important parameters in PSP, which are the number of employees, the number of tasks, and the number of employee skills. To capture more features of the realistic PSP, the MODPSP instances generated here differ from the static ones of [3] in the following aspects: (1) The task effort uncertainties and three kinds of dynamic events (new task arrivals, employee leaves, and employee returns), which often occur during the project execution, are incorporated. (2) The maximum headcount of each task, the skill level of each employee, part-time jobs, and overtime working of employees are all taken into account.

The 18 dynamic instances derived from benchmark 4 in [3] contain different software projects. The total number of different skills required by the project is 10 in each instance, and each task requires five skills randomly selected from them. The number of employees can be 5, 10, or 15, and the number of skills possessed by each employee ranges from 4 to 5, or from 6 to 7. 20 percent of the employees do part-time jobs, whose maximum dedications are uniformly generated from [0.5, 1) at random; another 20 percent can work overtime, whose maximum dedications are in the interval

(1, 1.5]; and the maximum dedication of each remaining employee is set to 1.0 (full time). According to [3], the normal monthly salary of each employee is sampled from a normal distribution with the mean of 10,000 and standard deviation of 1,000. The overtime salary is set to be the normal monthly salary multiplied by 3. If an employee has one skill, the proficiency score is sampled uniformly from (0, 5] at random, otherwise it is set to 0.

At the initial time, the number of tasks can be 10, 20, or 30. Then it is assumed that 10 new tasks arrive one by one following a Poisson distribution. We suppose the mean time between task arrivals is 1 month. We assume 20 percent of the new tasks are urgent, and the remaining 80 percent are regular. The simulation continues until all the original and new tasks have finished. Variances of task efforts are assumed to follow a normal distribution. Each task effort is assigned different values of mean and standard deviation, which vary uniformly in [8], [12] and [4], [6] (unit: person-month), respectively. These values are chosen such that on average, the mean of a task effort is 10 and the standard deviation is 5 [3]. The TPG for the initial tasks is generated using the method in [3]. For a newly arrived task, the precedence of the urgent or the regular one is inserted preceding or succeeding a randomly selected unfinished task, respectively.

During the project execution, employee leaves and returns are assumed to follow a Possion distribution. As indicated in Section 3.1, each employee is assigned different mean time between his/her leaves and mean time to his/her return, which vary uniformly in [11], [13] and [0.4, 0.6] (unit: month), respectively. These values are chosen such that on average, an employee is available for 11.5 months per year, and then asks for a leave of 0.5 months. Hence, an employee's availability is about 95.83 percent.

The 18 MODPSP instances randomly generated according to the above principles are named as sT#1_dT#2_E#3_SK#4-#5, where sT#1 means the number of initial static tasks, dT#2 means the number of dynamically arriving tasks, E#3 means the total number of employees, and SK#4-#5 means each employee has #4 to #5 skills. For example, sT30_dT10_E15_SK6-7 denotes that there are 30 tasks in the project initially, then 10 new tasks arrive one by one dynamically, and there are in total 15 employees, each of whom has six to seven skills.

Additionally, three real-world instances (named Real_1, Real_2 and Real_3) derived from business software construction projects for a departmental store [10] were also used in our experiments. Since these real instances are originally static PSPs, uncertainties and dynamic events are introduced to transfer them into the dynamic instances. For task effort uncertainties, the original task effort in the static real instances is regarded as the initially estimated effort, and is set as the mean value of the normal distribution which each task effort follows, and the standard deviation is assumed to be 10 percent of the mean value. The three kinds of dynamic events occur in the same way as that described in the above 18 randomly generated instances. In addition, the maximum dedication e_i^{maxded} in our model was calculated from the real instances as follows:

$$e_i^{maxded} = \frac{\text{maximum possible working hours per month}}{\text{legal normal working hours per month}}.$$

TABLE 4
Parameter Settings of the Rescheduling Method

Population size of dε-MOEA	100
Chromosome	Binary encoding, 3 bits for each $x_{ij}(t_l)$, i.e. nb = 3
Crossover possibility	0.9
Mutation possibility	$1/L$, where L is the chromosome length
maximum number of objective vector evaluations	10,000

When evaluating the cost in the procedure shown in Fig. 5, the basic salary was incorporated as in [10].

In total, there are 21 test instances used in our experiments, which were performed on a personal computer with Intel core i5, 3.2 GHz CPU and 4 GB RAM.

We do not currently have access to real world software project data containing information about their dynamic and uncertain events. The simulation nature and the lack of empirical validation with data of completely real nature is a threat to validity of this study. In order to mitigate this threat, we used several simulated software projects containing different numbers of tasks, employees, skills, dynamic events and uncertainties. We have also used three real world software projects with simulated dynamic and uncertain events. Once real world data with known dynamic and uncertain events become available for an empirical study, further analyses should be performed.

6.2 Parameter Settings

Parameter settings of our rescheduling method, dε-MOEA, in all the experiments are given in Table 4. In each independent run, the algorithm stops after 10,000 objective vector evaluations. In the decision making procedure, the pairwise comparison matrix for the four objectives was assumed to be

$$\mathrm{C}_1 = \left(c_{ij}\right)_{4\times 4} = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1/2 & 1/2 & 1 & 1 \\ 1/2 & 1/2 & 1 & 1 \end{bmatrix}.$$

Hence, the corresponding weight vector is $\mathrm{w} = (w_i)_{4\times 1} = [0.3333\ 0.3333\ 0.1667\ 0.1667]^T$.

6.3 Research Questions

The research questions (RQ) that our experimental studies aim to investigate are as follows:

RQ1. Is our initial proactive scheduling effective in improving the schedule robustness to task effort uncertainties by simultaneously considering the "*robustness*" objective and two efficiency objectives?

RQ2. Does our rescheduling method dε-MOEA improve the project efficiency significantly compared to a heuristic dynamic scheduling method? Is the project efficiency sensitive to task effort variances when using our rescheduling method dε-MOEA?

RQ3. Are the strategies designed in dε-MOEA effective compared to other MOEA-based rescheduling methods? These strategies include the dynamic optimization mechanism, introduction of the robustness and stability objectives, and heuristic initialization strategies.

TABLE 5
Statistical Tests of the Metric *C* between ε-MOEA-r and ε-MOEA-d for the 21 MODPSP Instances at the Initial Time

Instance		sT10_dT10_ E5_SK4–5	sT10_dT10_ E10_SK4–5	sT10_dT10_ E15_SK4–5	sT10_dT10_ E5_SK6–7	sT10_dT10_ E10_SK6–7	sT10_dT10_ E15_SK6–7
C(ε-MOEA-r, ε-MOEA-d) versus *C*(ε-MOEA-d, ε-MOEA-r)	*p*-value	0.0039	1.64E-6	0.3309	3.30E-6	1.13E-6	7.99E-6
	sign	+	+	=	+	+	+
Instance		sT20_dT10_ E5_SK4–5	sT20_dT10_ E10_SK4–5	sT20_dT10_ E5_SK4–5	sT20_dT10_ E5_SK6–7	sT20_dT10_ E10_SK6–7	sT20_dT10_ E15_SK6–7
C(ε-MOEA-r, ε-MOEA-d) versus *C*(ε-MOEA-d, ε-MOEA-r)	*p*-value	8.18E-4	0.8865	0.0501	4.58E-8	3.55E-6	5.34E-4
	sign	+	=	=	+	+	+
Instance		sT30_dT10_ E5_SK4–5	sT30_dT10_ E10_SK4–5	sT30_dT10_ E15_SK4–5	sT30_dT10_ E5_SK6–7	sT30_dT10_ E10_SK6–7	sT30_dT10_ E15_SK6-7
C(ε-MOEA-r, ε-MOEA-d) versus *C*(ε-MOEA-d, ε-MOEA-r)	*p*-value	0.0202	0.1671	0.0079	4.94E-8	6.37E-6	5.45E-6
	sign	+	=	=	+	+	+
Instance		Real_1	Real_2	Real_3			
C(ε-MOEA-r, ε-MOEA-d) versus *C*(ε-MOEA-d, ε-MOEA-r)	*p*-value	0.0361	2.86E-6	0.0017			
	sign	+	+	+			

The sign of '+/−/ =' in A versus B indicates that according to the metric C*, algorithm A is significantly better than B, significantly worse than B, or there is no significant difference between A and B based on the Wilcoxon rank sum test with the significance level of 0.05.*

RQ4. What insights into trade-offs among objectives can be found in the Pareto fronts of software projects?

6.4 RQ1: Influence of the Robustness Objective on the Initial Proactive Scheduling

This section aims to validate the effectiveness of the initial proactive scheduling in improving the schedule robustness to task effort uncertainties (the dynamic events occurring during the project execution are not considered here). Performance comparisons were done between our robust method (three objectives of $duration_I$, $cost_I$, and *robustness* were considered simultaneously, and it is called ε-MOEA-r), and the method where only two efficiency objectives ($duration_I$ and $cost_I$) were considered (it is called ε-MOEA-d). Aiming to compare the two methods within a multi-objective framework, the following steps were performed at the initial time of each of the 21 MODPSP instances:

Step i. Two methods were applied, and two non-dominated solution sets were produced, respectively. Then the value of the objective "*robustness*" was calculated for the two methods using the same sampled efforts (100 task effort scenarios were sampled at random here), despite the fact that only one of them was optimizing this objective. When comparing the two methods in terms of Pareto domination, "*robustness*" was also considered.

Step ii. The non-dominated sets of the two methods were compared using the set cover metric *C* [39], which is defined as follows: suppose X_1, X_2 are two solution sets. The metric *C* maps the ordered pair (X_1, X_2) into the interval $[0, 1]$:

$$C(X_1, X_2) = \frac{|\{\mathrm{x}_2 \in X_2; \exists \mathrm{x}_1 \in X_1 : \mathrm{x}_1 \prec \mathrm{x}_2 \mathrm{or} \mathrm{F}(\mathrm{x}_1) = \mathrm{F}(\mathrm{x}_2)\}|}{|X_2|}, \tag{24}$$

where $\mathrm{x}_1 \prec \mathrm{x}_2$ means solution x_1 Pareto dominates x_2, and F is the objective vector. The metric *C* gives a comparison of two sets based on their domination or equality to each other. $C(X_1, X_2) > C(X_2, X_1)$ indicates that X_1 is better than X_2 in terms of the metric *C*.

Step iii. In order to check the overall performance improvement (or deterioration) on individual objectives by using "*robustness*" as one of the multiple objectives, the non-dominated solutions of ε-MOEA-r were averaged along each of the three objectives, respectively, and also for ε-MOEA-d. The quantitative improvement (or deterioration) of ε-MOEA-r over ε-MOEA-d on each objective is calculated as follows:

$$Imp_i(t_0) = -\frac{\left(Avg\_f_i^{\varepsilon-\mathrm{MOEA-r}} - Avg\_f_i^{\varepsilon-\mathrm{MOEA-d}}\right)}{Avg\_f_i^{\varepsilon-\mathrm{MOEA-d}}} \times 100\%, \quad i = 1, 2, 3 \tag{25}$$

where $Avg\_f_i^{\varepsilon-\mathrm{MOEA-r}}$ and $Avg\_f_i^{\varepsilon-\mathrm{MOEA-d}}$ represent the average of the non-dominated solutions (i.e., the overall performance) obtained by ε-MOEA-r and ε-MOEA-d on the objective f_i, respectively. Since all objectives were to be minimized, we used a negative sign in (25).

30 independent runs of both methods were replicated following the above experimental procedure on each problem instance. To significantly compare the metric *C* between ε-MOEA-r and ε-MOEA-d on the 21 instances, Wilcoxon rank sum tests with the significance level of 0.05 were employed. The results are listed in Table 5. It can be seen that *C*(ε-MOEA-r, ε-MOEA-d) is significantly better than *C* (ε-MOEA-d, ε-MOEA-r) in 100 percent of the real-world, and 72.22 percent of the random instances, respectively, and there is no significant difference between them in the remaining 27.78 percent of the random instances, which indicates that the convergence performance of our robust method ε-MOEA-r is better than or at least no worse than ε-MOEA-d in terms of Pareto domination.

The overall performance improvement (or deterioration) of ε-MOEA-r over ε-MOEA-d on each objective was averaged over 30 runs and listed in Table 6. Wilcoxon rank sum tests with the significance level of 0.05 were employed to significantly compare the overall performance on each objective obtained by each of the two algorithms, and the results are

TABLE 6
Performance Improvements (or Deteriorations) of ε-MOEA-r over ε-MOEA-d and Statistical Tests of the Overall Performance on Each Objective on the 21 MODPSP Instances at the Initial Time

	Instance	$duration_I$	$cost_I$	*robustness*	Instance	$duration_I$	$cost_I$	*robustness*
ε-MOEA-r versus ε-MOEA-d	sT10_dT10_ E5_SK4-5	−1.63 percent (0.096 =)	**0.76** percent (0.15 =)	**4.05** percent (0.022 +)	sT10_dT10_ E10_SK4-5	−2.98 percent (0.22 =)	−2.29 percent (0.15 =)	**5.11** percent (0.022 +)
	sT10_dT10_ E15_SK4-5	**3.29** percent (0.0017 +)	−1.42 percent (0.16 =)	**2.56** percent (0.08 =)	sT10_dT10_ E5_SK6-7	−4.46 percent (0.12 =)	−0.53 percent (0.21 =)	**6.42** percent (0.003 +)
	sT10_dT10_ E10_SK6-7	**2.33** percent (6.91E−4 +)	−0.94 percent (0.14 =)	**7.88** percent (7.20E−5 +)	sT10_dT10_ E15_SK6-7	−6.13 percent (0.0087 −)	**0.33** percent (0.15 =)	**11.36** percent (0.023 +)
	sT20_dT10_ E5_SK4-5	−1.79 percent (0.080 =)	**0.75** percent (0.52 =)	**13.42** percent (2.22E−5 +)	sT20_dT10_ E10_SK4-5	−4.11 percent (8.66E−5 −)	−1.03 percent (0.082 =)	**17.06** percent (8.88E−5 +)
	sT20_dT10_ E15_SK4-5	−6.25 percent (0.047 −)	−1.51 percent (0.59 =)	**16.70** percent (3.83E−5 +)	sT20_dT10_ E5_SK6-7	−2.15 percent (0.085 =)	−1.14 percent (0.064 =)	**9.07** percent (0.0076 +)
	sT20_dT10_ E10_SK6-7	−1.11 percent (0.058 =)	−1.14 percent (0.59 =)	**3.62** percent (1.47E−7 +)	sT20_dT10_ E15_SK6-7	**3.72** percent (4.46E−4 +)	**0.53** percent (0.75 =)	**5.84** percent (4.12E−6 +)
	sT30_dT10_ E5_SK4-5	−0.58 percent (0.064 =)	**1.31** percent (0.077 =)	**9.53** percent (1.17E−4 +)	sT30_dT10_ E10_SK4-5	−7.86 percent (8.20E−7 −)	−7.28 percent (0.66 =)	**32.25** percent (1.96E-10 +)
	sT30_dT10_ E15_SK4-5	−1.31 percent (0.21 =)	−0.17 percent (0.060 =)	**6.19** percent (5.27E−5 +)	sT30_dT10_ E5_SK6-7	−6.61 percent (0.0023 −)	−0.94 percent (0.52 =)	**17.48** percent (2.60E−5 +)
	sT30_dT10_ E10_SK6-7	−6.81 percent (4.22E−4 −)	−2.11 percent (0.43 =)	**22.82** percent (3.96E−8 +)	sT30_dT10_ E15_SK6-7	−8.67 percent (6.36E−5 =)	-0.28 percent (0.84 =)	**21.97** percent (2.83E−8 +)
	Real_1	−6.25 percent (2.60E−8 −)	−4.14 percent (4.12E−8 −)	**27.84** percent (1.61E−10 +)	Real_2	−0.31 percent (0.70 =)	**1.82** percent (0.028 +)	**9.29** percent (0.0018 +)
	Real_3	−1.56 percent (0.36 =)	1.53 percent (0.0468 +)	**2.13** percent (0.029 +)				

The positive value means improvement and is in bold. The negative value means deterioration. The sign of '+/−/ =' in A versus B indicates that according to the overall performance on each objective, algorithm A is significantly better than B, significantly worse than B, or there is no significant difference between A and B based on the Wilcoxon rank sum test with the significance level of 0.05. The values in the parentheses are p*-values obtained from Wilcoxon rank sum tests.*

also shown in Table 6. It can be seen from statistical results that compared to ε-MOEA-d, ε-MOEA-r improves the robustness significantly in 17 of the 18 random instances and all the three real instances, while only deteriorates the efficiency objective significantly (mainly the duration) in six of the 18 random instances and one of the three real instances. From the results of overall performance improvement, the improvement in robustness is much more than the deterioration in efficiency, which suggests that if the predictive schedules are generated by simultaneously considering robustness and efficiency, there will be a high chance of obtaining more robust schedules without seriously affecting efficiency. Note that this happens not only in the random instances, but also in the ones derived from real-world projects. Moreover, the better robustness performance obtained by ε-MOEA-r shows it can produce a set of trade-off schedules with lower duration delays and cost increases than ε-MOEA-d when facing task effort uncertainties, which suggests its ability to reduce the schedule sensitivity to uncertainties.

Take Real_3 as an example to illustrate the behaviors of different algorithms. The initially estimated efforts of the 12 initial static tasks are 3, 2, 2, 2, 3, 1, 4, 3, 2, 2, 2, and 3, and a disrupted task effort scenario is 2.84, 2.16, 1.76, 2.16, 3.36, 1.13, 3.31, 3.28, 2.13, 1.87, 2.35 and 2.72 respectively. Duration and cost in the initial ($duration_I$ and $cost_I$) and disrupted ($duration_q$ and $cost_q$) scenario of a randomly chosen schedule generated by ε-MOEA-d are shown in Table 7 and also for ε-MOEA-r. It can be seen that to get better robustness, the initial duration and cost of ε-MOEA-r are worse than those of ε-MOEA-d. However, when facing the same task effort disruption, the disrupted duration and cost for ε-MOEA-d becomes worse than both the initial and disrupted duration and cost for ε-MOEA-r, which illustrates that better robustness really compensates the worse initial cost and duration for ε-MOEA-r.

6.5 RQ2: Comparisons of the Proposed Rescheduling Method dε-MOEA against the Heuristic Dynamic Scheduling Method

This section compares dε-MOEA with a heuristic dynamic scheduling method (it is called h-method), which generates an initial schedule by the robust scheduling algorithm ε-MOEA-r (introduced in Section 6.3), combined with the decision making method described in Section 4.2.4. The h-method then makes a local adjustment to the original

TABLE 7
An Example of the Duration and Cost Variance Obtained by ε-MOEA-d and ε-MOEA-r in the Initial and Disrupted Scenario

	$duration_I$	$cost_I$	$duration_q$	$cost_q$	*robustness*
ε-MOEA-d	8.34	368583	8.81	380880	0.0563
ε-MOEA-r	8.41	370630	8.68	378275	0.0516

TABLE 8
Average Results, Percentages of the Performance Improvement, and the Statistical Test Results Obtained by Comparing dε-MOEA against h-Method

Performance values	Project duration	Project cost	Project duration	Project cost	Project duration	Project cost	Project duration	Project cost
Instance	sT10_dT10_E5_SK4–5		sT10_dT10_E10_SK4–5		sT10_dT10_E15_SK4–5		sT10_dT10_E5_SK6–7	
dε-MOEA	**108.4**	**3512722**	**71.0**	**2792531**	**57.1**	**2019370**	**145.9**	**3601760**
h-method	136.2	3862030	91.0	3148683	68.2	2280162	172.5	4072819
Improvement percentage	20.4 percent	9.0 percent	22.0 percent	11.3 percent	16.3 percent	11.4 percent	15.4 percent	11.6 percent
dε-MOEA versus h-method	3.02E-11+	3.02E-11+	3.61E-13+	8.73E-14+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+
Instance	sT10_dT10_E10_SK6–7		sT10_dT10_E15_SK6–7		sT20_dT10_E5_SK4–5		sT20_dT10_E10_SK4–5	
dε-MOEA	**65.3**	**2451803**	**60.1**	**2404944**	**161.9**	**6090925**	**65.9**	**3136695**
h-method	89.3	2799789	78.2	2738974	188.1	6619072	87.2	3391533
Improvement percentage	26.9 percent	12.4 percent	23.2 percent	12.2 percent	13.9 percent	8.0 percent	24.4 percent	7.5 percent
dε-MOEA versus h-method	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+
Instance	sT20_dT10_E15_SK4–5		sT20_dT10_E5_SK6–7		sT20_dT10_E10_SK6–7		sT20_dT10_E15_SK6–7	
dε-MOEA	**58.6**	**3096519**	**230.9**	**6955361**	**78.8**	**4466765**	**64.3**	**3594815**
h-method	71.3	3265817	293.8	7308816	108.3	5205138	87.2	3745180
Improvement percentage	17.8 percent	5.2 percent	21.4 percent	4.8 percent	27.2 percent	14.2 percent	26.3 percent	4.0 percent
dε-MOEA versus h-method	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+
Instance	sT30_dT10_E5_SK4–5		sT30_dT10_E10_SK4–5		sT30_dT10_E15_SK4–5		sT30_dT10_E5_SK6–7	
dε-MOEA	**120.4**	**5017805**	**82.4**	**5013772**	**69.6**	**4607639**	**196.6**	**7786837**
h-method	137.2	5392107	110.2	5242736	92.9	4838638	216.3	8150944
Improvement percentage	12.2 percent	6.9 percent	25.2 percent	4.4 percent	25.1 percent	4.8 percent	9.2 percent	4.5 percent
dε-MOEA versus h-method	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+
Instance	sT30_dT10_E10_SK6–7		sT30_dT10_E15_SK6–7		Real_1		Real_2	
dε-MOEA	**128.1**	**6887576**	**93.0**	**6094726**	**16.2**	**1330256**	**12.1**	**499891**
h-method	157.0	7571868	126.6	6317385	19.9	1451890	14.6	611864
Improvement percentage	18.4 percent	9.0 percent	26.6 percent	3.5 percent	18.6 percent	8.4 percent	17.1 percent	18.3 percent
dε-MOEA versus h-method	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+	3.02E-11+
Instance	Real_3							
dε-MOEA	**16.6**	**690717**						
h-method	20.3	856075						
Improvement percentage	18.2 percent	19.3 percent						
dε-MOEA versus h-method	3.02E-11+	3.02E-11+						

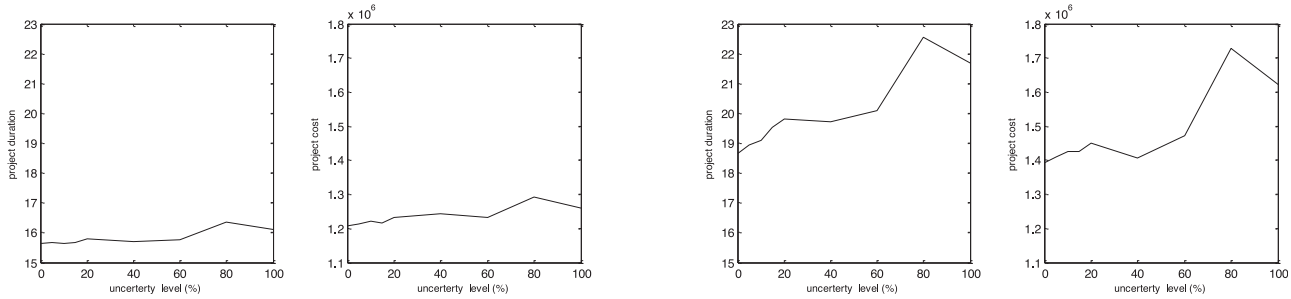
The better of the performance values on each instance are in bold. The unit of the project duration is month. The sign of '+/−/=' in A versus B indicates that according to the performance compared, algorithm A is significantly better than B, significantly worse than B, or there is no significant difference between A and B based on the Wilcoxon rank sum test with the significance level of 0.05. The values before the signs '+/−/=' are p*-values obtained from Wilcoxon rank sum tests.*

schedule based on a heuristic rule when a dynamic event occurs (these local adjustments are different strategies that could be adopted if no dynamic MOEA rescheduling was used). The heuristic rules used here are:

1) In the case that an employee leaves and returns, for each task in which the leaving employee is on, if the task becomes infeasible because the employee leaves, then the task is unprocessed and waits until the employee returns to be continued. Otherwise, if the task is still feasible, then the remaining employees still work on it and their dedications to the task are kept unchanged. For other tasks, they are performed according to the initial schedule.
2) If a newly arrived task is to be performed according to the TPG, then a number of available employees with higher proficiencies (measured by $e_{ij}^{Proficiency}$) will be assigned to it, simultaneously satisfying the task skill constraint. The number of selected employees is expected not to exceed the maximum headcount of the task. However, if the team size cannot be reduced to the limit without violating the task skill constraint, then the task headcount constraints can be relaxed.

Two performance measures were adopted in this section. One was the whole project duration (the elapsed time of finishing all the tasks that have ever been considered as part of the project), and the other was the whole project cost (the total expenses paid to the employees for completing the whole project). 30 independent runs on each MODPSP instance were performed using our method dε-MOEA, and also the h-method. Average results, percentages of the performance improvement of dε-MOEA over h-method, and the statistical results obtained by Wilcoxon rank sum tests with the significance level of 0.05 are listed in Table 8.

It can be observed that compared to h-method, dε-MOEA decreases the whole project duration and cost significantly on all instances. It improves the project efficiency to a large extent, which shows the distinct superiority of dε-MOEA over the heuristic dynamic scheduling when dealing with MODPSP, although the mean CPU time consumed by dε-MOEA at each scheduling point is much larger than that of the h-method (The smallest and largest mean execution time cost by dε-MOEA was 86.33 s (Real_1) and 431.68 s (sT30_dT10_E10_SK6-7), while those of h-method were only 0.0150 s (Real_3) and 0.0556 s (sT30_dT10_E5_SK6-7), respectively). However, compared to the project duration measured by months and the savings found by dε-MOEA,



(a) dε-MOEA (b) h-method

Fig. 6. Variations of the project duration and cost obtained by dε-MOEA and h-method with the task effort uncertainties.

the time cost of dε-MOEA is relatively small, and it is worth consuming the time to regenerate a schedule by dε-MOEA that can improve the project efficiency significantly.

The sensitivity analysis of the impact of task effort variances on the project efficiency is also performed on one real-world instance (Real_1). Here, the standard deviation of the normal distribution that task effort variances assumed to follow is set to be 5, 10, 15, 20, 40, 60, 80, and 100 percent of the mean value, respectively, which reflects the uncertainty level. Fig. 6 gives the variations of the project duration and cost obtained by dε-MOEA and h-method with the uncertainty in the task effort estimation (30 replications of either method are performed under each uncertainty level and the average value is computed, respectively). It can be seen that as the uncertainty level increases, the project duration and cost also increase because the project suffers from such effort variations. However, the increment of project duration and cost produced by dε-MOEA (Fig. 6a) is much smaller than that obtained by h-method (Fig. 6b), which indicates that our method dε-MOEA is much less sensitive against such task effort uncertainties.

6.6 RQ3: Validating the Effectiveness of Strategies Designed in dε-MOEA

6.6.1 Introduction to the Compared Methods

In this section, the proposed rescheduling method dε-MOEA was compared to the other four rescheduling methods which are listed as follows:

i) *dCOEA.* To validate the effectiveness of the dynamic optimization mechanism incorporated in dε-MOEA, it was compared to a state-of-the-art dynamic MOEA called dCOEA [40]. At each scheduling point, each subpopulation in dCOEA played a role in searching the dedications of one available employee to all available tasks. In dCOEA, two strategies were specifically designed for dynamic optimization: (1) when changes occur, diversity in each subpopulation was introduced via stochastic competitors; and (2) to exploit useful information about the current archive, a temporal memory was used to handle outdated archived solutions. The chromosome representations and variation operators in dCOEA were the same as those in dε-MOEA. The parameter settings of dCOEA were: the subpopulation size was 10, the maximum archive size was 100, SC_{ratio} was 0.5, R_{size} was 5, and C_{freq} was 10, which were the same as recommended by [40]. Other parameters such as the crossover and mutation probabilities were the same as those in dε-MOEA.

ii) *dε-MOEA-Deterministic.* To demonstrate the superiority of considering project duration, cost, robustness and stability simultaneously and incorporating the heuristic initialization, dε-MOEA was compared to dε-MOEA-Deterministic, which is an ε-MOEA-based complete rescheduling method [15] that regenerates a new schedule from scratch and does not consider task effort uncertainties and project stability. At each scheduling point t_l, only the project duration and cost in the initial scenario are considered. Meanwhile, the initial population is entirely generated at random.

iii) *dε-MOEA-No-Sta.* To study the impact of the stability objective, dε-MOEA was compared to an ε-MOEA-based rescheduling method without considering stability called dε-MOEA-No-Sta. This method is different from dε-MOEA in that only three objectives ($duration_I$, $cost_I$ and $robustness$) are optimized simultaneously at each scheduling point (heuristic initialization is adopted). From this group of comparisons, a software manager can gain insight into how the initial duration, cost and robustness would be affected by considering stability and whether the human allocation and dedication changes would become smaller at different scheduling points.

iv) *dε-MOEA-No-HI.* To study the influence of heuristic initialization strategies, dε-MOEA was compared to an ε-MOEA-based rescheduling method which just adopted random initialization. This method is different from dε-MOEA in that the initial population is generated at random at each scheduling point (four objectives are considered simultaneously). This group of experiments can provide a software manager with a better understanding of whether it would be helpful to utilize dynamic features of a problem and exploit the previous schedule information when re-planning a schedule.

The parameter settings of dε-MOEA-Deterministic, dε-MOEA-No-Sta, and dε-MOEA-No-HI are the same as those of dε-MOEA, which are given in Table 4. Note that all algorithms stop after 10,000 objective vector evaluations in one run.

6.6.2 Performance Measures

It is desirable for an algorithm to provide a software manager with a set of non-dominated solutions with good

convergence to the reference Pareto optimal front, and also with a uniform (in most cases) distribution and a wide spread over the Pareto front. In this way, the software manager can get a full picture of various trade-offs among the project duration, cost, robustness and stability, which is very helpful for him/her to understand more about the problem so that he/she can make an informed choice or revise the schedule already planned by himself/herself according to the requirement of the project.

In this paper, four popular metrics are employed to evaluate the performance of the five MOEA-based rescheduling methods. The first one is the *hypervolume ratio* (*HVR*) [41]. The hypervolume metric *HV* measures the size of the objective space dominated by the obtained non-dominated front PF_{known} [42], and *HVR* is the ratio of *HV* and the hypervolume of the reference Pareto front $PF_{ref.}$ A larger *HVR* value indicates a better convergence and a wider spread of the obtained non-dominated front. The second one is the *Generational Distance* (*GD*), which measures how far PF_{known} is from PF_{ref} [43]. A small *GD* indicates the obtained solutions are close to the reference Pareto front, which means a good convergence performance. The weakness of *GD* is that it does not take the spread of solutions into account, hence a set of solutions which gather around a small region near the reference Pareto front may also get a good *GD* value. The third one is a distribution performance metric called *Spacing*, which measures the distance variations of neighbouring vectors in PF_{known} [44]. The smaller *Spacing* is, the better the distribution uniformity of PF_{known} is. The fourth one is *Spread*, which measures the extent of spread achieved by the obtained solutions and the uniformity in the distribution of PF_{known}. The definition of *Spread* in [45] was used for bi-objective problems. As for problems with three or more objectives, we propose a modified *Spread* given in (26):

$$Spread = \frac{\sum_{j=1}^{N\_o} df_j + \sum_{i=1}^{n_{PF}} \left| d'_i - \overline{d'} \right|}{\sum_{j=1}^{N\_o} df_j + n_{PF} \cdot \overline{d'}}, \tag{26}$$

where N_o is the number of objectives, df_j is the Euclidean distance between the best solution on the jth objective and its nearest solution in PF_{known}, n_{PF} is the number of vectors in PF_{known}, d'_i is the Euclidean distance from the ith vector of PF_{known} to its nearest neighbour in PF_{known}, and $\overline{d'}$ is the mean of all d'_i. A wide and uniform spread of solutions in PF_{known} will result in a small value of *Spread*.

No matter how uniformly the solutions distribute or how widely the range of objective values covers, if the obtained solution set is far from the reference Pareto front, the algorithm is not very useful because some of the project cost, duration, robustness and stability are poor. Thus, the convergence performance (*HVR* and *GD*) of an algorithm should be considered first by a software manager when choosing an algorithm to use. For two algorithms with comparable convergence, the one with a better distribution (*Spacing*) and spread (*Spread*) is preferred.

Because the true Pareto front at each scheduling point is unknown in MODPSP, PF_{ref} is obtained in our work by merging the solutions found during all the independent runs using all the five methods, and then obtaining the non-dominated solutions from them. The reference point in *HVR* is formed by the worst objective values observed in all optimization runs.

Due to the space limitation, the procedure of comparing dε-MOEA to other MOEA-based rescheduling methods is presented in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2015.2512266, in detail. To compare the five methods in terms of the overall performance across different scheduling points and runs on each instance, Wilcoxon rank sum tests with the significance level of 0.05 are employed. The statistical test results are listed in Table B.1 in Appendix B, available in the online supplemental material. The overall performance improvement (or deterioration) and the statistical test results of dε-MOEA over the other four methods on each objective on the 21 instances are listed in Table B.2 in Appendix B, available in the online supplemental material.

6.6.3 Comparisons to dCOEA

In order to understand the impact of different dynamic MOEAs on the performance of MODPSP, we also applied dCOEA to find the Pareto front at each scheduling point. Table 9 summarizes the statistical test results of our method dε-MOEA versus the other four methods.

It can be seen that in terms of the convergence metrics *HVR* and *GD*, dε-MOEA is significantly better than dCOEA in all cases. It maintains a comparable spread performance to dCOEA, where the *Spread* values of dε-MOEA are significantly better than dCOEA in 83 percent of the 18 random instances and one in three real-world instances, respectively. As to the distribution performance, dε-MOEA is comparable to dCOEA on the real-world instances, while a bit worse than dCOEA on the random instances since its *Spacing* values are significantly worse than dCOEA in 33 percent of the 18 random instances. One possible reason for this is that dCOEA is a coevolutionary algorithm known to be good at maintaining a diverse set of solutions.

As mentioned before, convergence performance is the most important factor that a software manager should take into account when evaluating an algorithm. The poor convergence performance of dCOEA in our experiments indicates that the dynamic optimization strategies it adopts may not be suitable for solving MODPSP (dCOEA was tested only in dynamic multi-objective function optimization in [40]). Other policies, such as the heuristic initialization strategies designed in this paper which can utilize dynamic features of MODPSP, should be introduced.

6.6.4 Comparisons to dε-MOEA-Deterministic

With the aim to observe the consequence caused by not considering uncertainties and system stability when rescheduling from scratch in MODPSP, dε-MOEA was compared to dε-MOEA-Deterministic, which only cares about the project duration and cost in the initial scenario and generates the initial population at random. It can be seen from Table 9 that considering the convergence metrics *HVR* and *GD*, dε-MOEA is significantly better than dε-MOEA-Deterministic in all cases. As for the *Spread* metric, dε-MOEA behaves better because it is significantly better than dε-MOEA-Deterministic in 61 percent of the random instances and

TABLE 9
Comparison Results Summarized from Table B.I (The Percentage of the 18 Random Instances and Three Real-World Instances for the Statistical Test Results of Dε-MOEA versus the Other Four Methods, where the Sign of '+/−/ = ' in A versus B Indicates that According to the Metric Considered, Algorithm A Is Significantly Better than B, Significantly Worse than B, or There Is No Significant Difference between A and B Based on the Wilcoxon Signed-Rank Test with the Significance Level of 0.05)

Random Instances												
	HVR			*GD*			*Spacing*			*Spread*		
dε-MOEA versus	+	=	−	+	=	−	+	=	−	+	=	−
dCOEA	100 percent	0	0	100 percent	0	0	28 percent	39 percent	33 percent	83 percent	17 percent	0
dε-MOEA versus dε-MOEA-Deterministic	+	=	−	+	=	−	+	=	−	+	=	−
	100 percent	0	0	100 percent	0	0	11 percent	39 percent	50 percent	61 percent	33 percent	6 percent
dε-MOEA versus	+	=	−	+	=	−	+	=	−	+	=	−
dε-MOEA-No-Sta	72 percent	28 percent	0	28 percent	72 percent	0	50 percent	50 percent	0 percent	67 percent	33 percent	0
dε-MOEA versus	+	=	−	+	=	−	+	=	−	+	=	−
dε-MOEA-No-HI	100 percent	0	0	100 percent	0	0	22 percent	45 percent	33 percent	11 percent	39 percent	50 percent
Real-world Instances												
	HVR			*GD*			*Spacing*			*Spread*		
dε-MOEA versus	+	=	−	+	=	−	+	=	−	+	=	−
dCOEA	100 percent	0	0	100 percent	0	0	33 percent	67 percent	0	33 percent	67 percent	0
dε-MOEA versus dε-MOEA-Deterministic	+	=	−	+	=	−	+	=	−	+	=	−
	100 percent	0	0	100 percent	0	0	0	33 percent	67 percent	100 percent	0	0
dε-MOEA versus	+	=	−	+	=	−	+	=	−	+	=	−
dε-MOEA-No-Sta	100 percent	0	0	33 percent	67 percent	0	33 percent	67 percent	0	0 percent	100 percent	0
dε-MOEA versus	+	=	−	+	=	−	+	=	−	+	=	−
dε-MOEA-No-HI	100 percent	0	0	100 percent	0	0	33	67 percent	0	34 percent	33 percent	33 percent

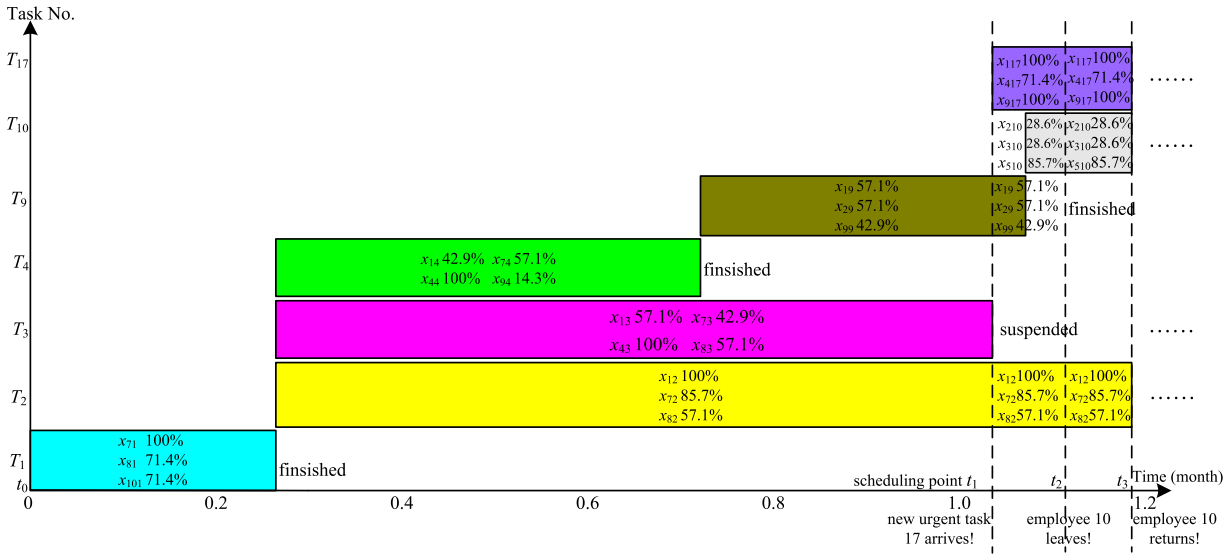
100 percent of the real-world instances. However, in terms of *Spacing*, dε-MOEA-Deterministic behaves better, since it is significantly better than dε-MOEA in 50 and 67 percent of the random and real-world instances, respectively. The possible reason is that the initial population are generated at random in dε-MOEA-Deterministic, which is helpful in increasing the diversity of solutions.

Besides, it can be found from Table B.2, available in the online supplemental material, that compared to dε-MOEA-Deterministic, dε-MOEA improves the overall performance on *robustness* and *stability* significantly in all cases, while it degrades $duration_I$ and (or) $cost_I$ sometimes. However, the improvements in *robustness* and *stability* are much more than the deterioration in the initial efficiency, which suggests that if a software manager reschedules by simultaneously considering duration, cost, robustness and stability, and also taking the dynamic event features and previous schedule information into account, he/she will have a higher chance of obtaining more robust and stable solutions without severely affecting the initial efficiency.

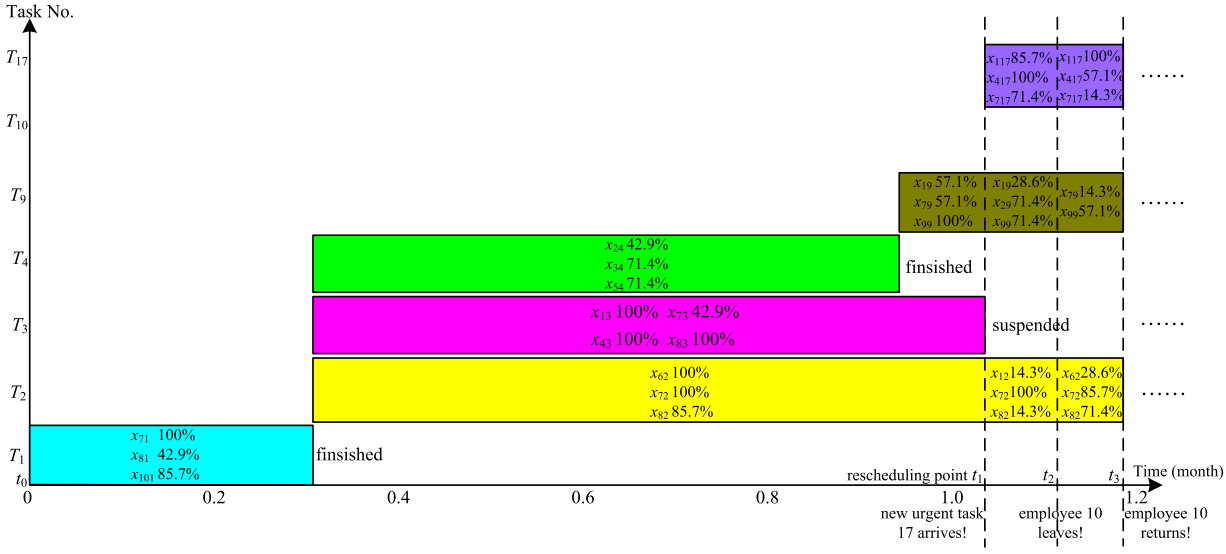
To further present the advantages of dε-MOEA over dε-MOEA-Deterministic, we plotted a section of the schedule Gantt charts produced by the two methods on one real-world instance (Real_2), respectively, which are given in Fig. 7. Since stability is not taken into account by dε-MOEA-Deterministic, it is possible that a group of employees different from the previous ones are assigned to the same task when rescheduling, and the dedication of an employee to a task fluctuates a lot. For example, in Fig. 7b, task T_9 is scheduled to be performed by employees e_1, e_7, e_9 at the initial time t_0, by e_1, e_2, e_9 at the scheduling point t_1, and by e_7, e_9 at t_2. Although e_9 is assigned to T_9 all the time, his/her dedication changes a lot. This will induce the system instability and lack of continuity, which is undesirable for any real-world software project. In contrast, by considering stability, the schedule in Fig. 7a produced by dε-MOEA is more stable with only small adjustments in a few dedications, and the group of employees assigned to tasks T_2, T_9, T_{10}, T_{17} are kept unchanged at different scheduling points as shown in Fig. 7a. Furthermore, since robustness is not considered in dε-MOEA-Deterministic either, its schedule may behave worse when facing task effort disturbances. For example, in Fig. 7b, the durations of tasks T_1 and T_4 were longer than those built by dε-MOEA in Fig. 7a. Besides, T_3 was suspended when the new urgent task T_{17} arrived (the precedence of T_{17} was higher than that of T_3) and would continue until T_{17} finished.

6.6.5 The Influence of the Stability Objective

To study the impact that the stability objective has on the performance of the MOEA-based rescheduling methods, dε-MOEA was compared to dε-MOEA-No-Sta which did not take the stability objective into account. It can be seen from Table 9 that considering the convergence metric *HVR*, dε-MOEA is significantly better than dε-MOEA-No-Sta in 72 and 100 percent of the random and real-world instances, respectively, which indicates that compared to dε-MOEA-No-Sta, dε-MOEA can provide the software manager with a wider spread of non-dominated solutions that are close to the reference Pareto front. As for *GD*, the differences between the two methods are not large: there is no significant difference between them in 72 and 67 percent of the random and real-world instances, respectively. The *Spread*



(a) A section of the schedule Gantt chart found by dε-MOEA at different scheduling points



(b) A section of the schedule Gantt chart found by dε-MOEA-Deterministic at different scheduling points

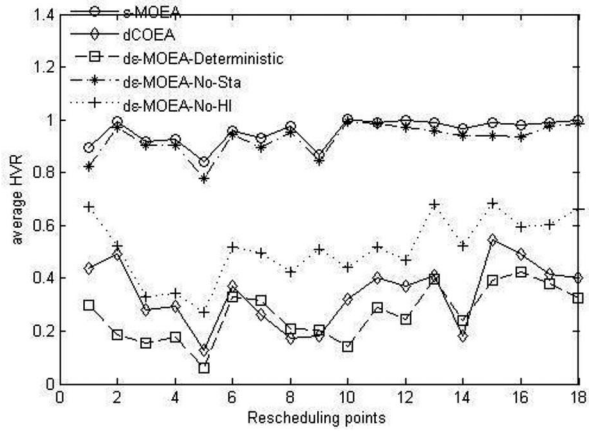
Fig. 7. Comparisons of the schedule Gantt charts produced by dε-MOEA and dε-MOEA-Deterministic in the real-world instance Real_2 (x_{ij} denotes the dedication of employee e_i to task T_j in the corresponding schedule).

values produced by dε-MOEA are better than or comparable to dε-MOEA-No-Sta in all the instances, and similar results can be obtained for the *Spacing* metric. The reason for dε-MOEA-No-Sta having a relatively good performance on *GD*, but not so good on *HVR* and *Spread* is that it can find a set of solutions close to the reference Pareto front, but they just gather around a small region (with good values of $duration_I$, $cost_I$ and *robustness*, but bad *stability*), so the spread of its solutions is not wide.

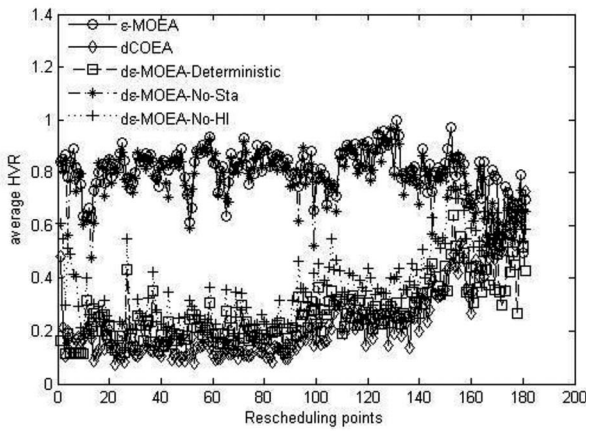
Besides, it can be found from Table B.2, available in the online supplemental material, that compared to dε-MOEA-No-Sta, dε-MOEA improves the system stability significantly with a small sacrifice in the initial efficiency and (or) robustness. This result is very practical for a software manager since stability is an important factor in the real-world software project.

6.6.6 The Influence of Heuristic Initialization Strategies

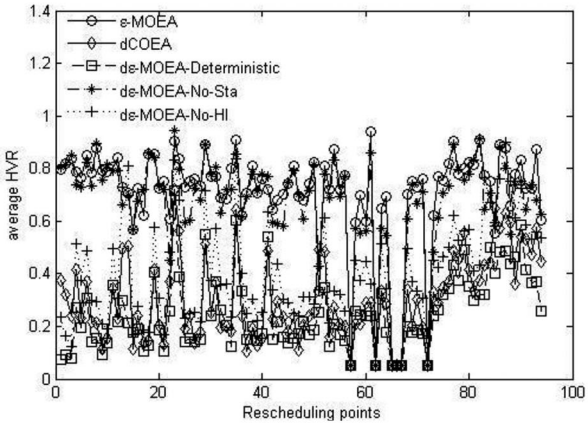
To study the impact that the heuristic initialization strategies have on the performance of the rescheduling method, dε-MOEA was compared to dε-MOEA-No-HI. It can be seen from Table 9 that considering the convergence metrics *HVR* and *GD*, dε-MOEA is significantly better than dε-MOEA-No-HI in all cases, which indicates that the combined use of dynamic features and history information in initialization can help improve the convergence performance of the MOEA-based rescheduling method a lot. Thus, when rescheduling, it is better for a software manager to take both the dynamic event features and previous schedule information into account. As for the *Spacing* metric, dε-MOEA outperforms or is comparable to dε-MOEA-No-HI in all the real-world instances, but it is significantly worse than dε-MOEA-No-HI in 33 percent of



(a) *HVR* comparisons on the instance Real_2 (with the best mean value of *HVR* obtained by dε-MOEA)



(b) *HVR* comparisons on the instance sT20_dT10_E10_SK6-7(with the medium mean value of HVR obtained by dε-MOEA)



(c) *HVR* comparisons on the instance sT10_dT10_E5_SK4-5 (with the worst mean value of *HVR* obtained by dε-MOEA)

Fig. 8. Average *HVR* comparisons of the five methods at each scheduling point on the MODPSP instance (*HVR* is to be maximized).

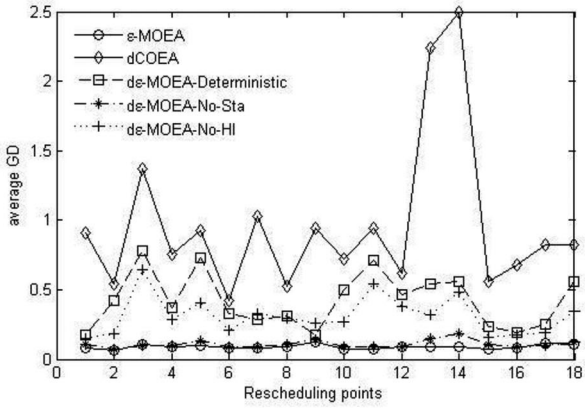
the random instances. Meanwhile, dε-MOEA-No-HI has better *Spread* performance as a whole since it is significantly better than dε-MOEA in 50 and 33 percent of the random and real-world instances, respectively. The reason is that dε-MOEA uses the history solution, the schedule repair solution and their variants as parts of the initial population, which can help speed up the convergence. However, this may limit the search space explored by the algorithm.

It can also be found from Table B.2, available in the online supplemental material, that compared to dε-MOEA-No-HI, dε-MOEA improves the overall performance on $cost_I$ and *stability* significantly in all cases, and improves $duration_I$ significantly in 16 of the 18 random instances and in all the 3 real instances, while it may degrade *robustness* in some instances. However, the improvements in $duration_I$, $cost_I$ and *stability* are much more than the deterioration in *robustness* (if any), which suggests that the incorporation of heuristic initialization is able to improve the efficiency and stability significantly with a small sacrifice in robustness.

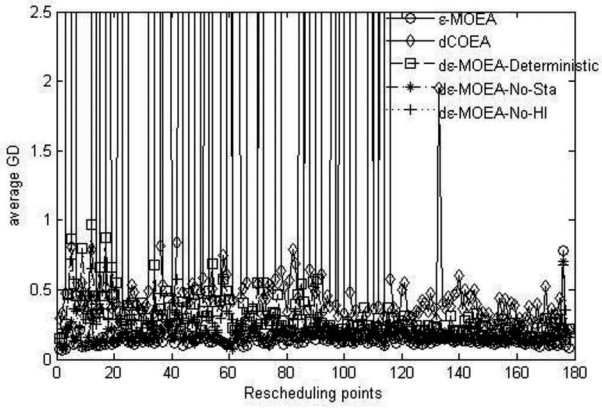
To further understand the advantages and disadvantages of the convergence performance of dε-MOEA over the other four methods, we plotted the average *HVR* and *GD* values over 30 independent runs across the scheduling points, which are shown in Figs. 8 and 9, respectively. Due to the space limitation, we just give curves on the instances with the best, medium and worst mean value of *HVR* or *GD* obtained by dε-MOEA. It can be seen that dε-MOEA can achieve the maximum *HVR* or the minimum *GD* value at most of the scheduling points. The convergence performance of dε-MOEA-No-Sta is close to that of dε-MOEA, while the other three methods are much worse.

6.7 RQ4: Pareto Fronts of the Evolved Schedules at Scheduling Points

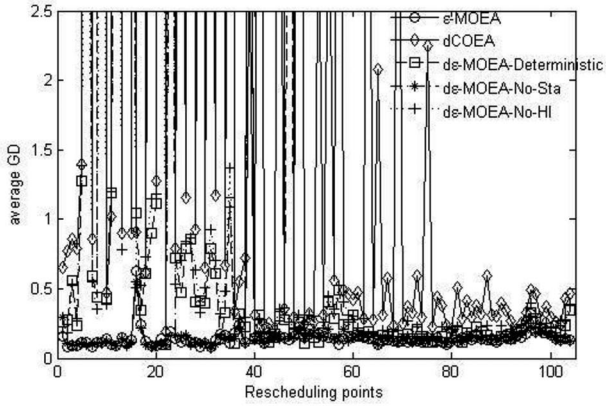
At each scheduling point, a set of non-dominated solutions were evolved by dε-MOEA. In order to demonstrate the trade-offs among these solutions which a software manager can utilize in balancing their choices when making a decision about the final schedule, one scheduling point on a real-world instance (Real_3) was selected arbitrarily and taken as an example. At $t_l = 2.6$ (month), the employee e_{10} leaves, and tasks T_4 to T_{14} are available. 31 independent runs of dε-MOEA were performed. With the aim of showing the sample median quality attained in multiple (31 here)



(a) *GD* comparisons on the instance Real_2 (with the best mean value of *GD* obtained by dε-MOEA)



(b) *GD* comparisons on the instance sT30_dT10_E15_SK4-5 (with the medium mean value of *GD* obtained by dε-MOEA)



(c) *GD* comparisons on the instance sT20_dT10_E5_SK4-5 (with the worst mean value of *GD* obtained by dε-MOEA)

Fig. 9. Average *GD* comparisons of the five methods at each scheduling point on the MODPSP instance (*GD* is to be minimized).

runs, the 50 percent -summary attainment surface (i.e., the 16th-summary attainment surface) [46] is obtained. A four objective problem requires 4D data to be represented. To visually investigate the resulting summary attainment surface, we give the slice plot in Fig. 10. The slice plot draws slices along the $duration_I$, $cost_I$ and *robustness* directions, and the colors on the slices are determined by the values on *stability*. As indicated in [46], the summary attainment surface emphasizes the distribution of the location achieved over multiple runs. Thus, it can be seen from Fig. 10 that the points on the 50 percent-summary attainment surface tend to crowd around the regions with small values on the objective $duration_I$, i.e., the density of such regions is much higher than others.

To inspect different trade-offs among the four objectives found by dε-MOEA in one run, one of the 31 Pareto fronts obtained from 31 runs of dε-MOEA was selected randomly. To visually investigate the Pareto front, we give the diagonal plot [47] in Fig. 11. The diagonal plot gives pairwise interactions among the four objective values on the Pareto front, where the axes of any plot can be obtained by finding the corresponding diagonal boxes and their ranges. For instance, the plot at the third row and fourth column has its vertical axis as *robustness* and horizontal axis as *stability*. Firstly, it can be observed from the figure $duration_I$ versus $cost_I$ that the two efficiency objectives are conflicting with each other, since a smaller $duration_I$ normally leads to a larger $cost_I$. Secondly, it can be seen from figures $cost_I$ versus *robustness* and $cost_I$ versus *stability* that the *robustness* or *stability* measure is slightly conflicting with the objective of

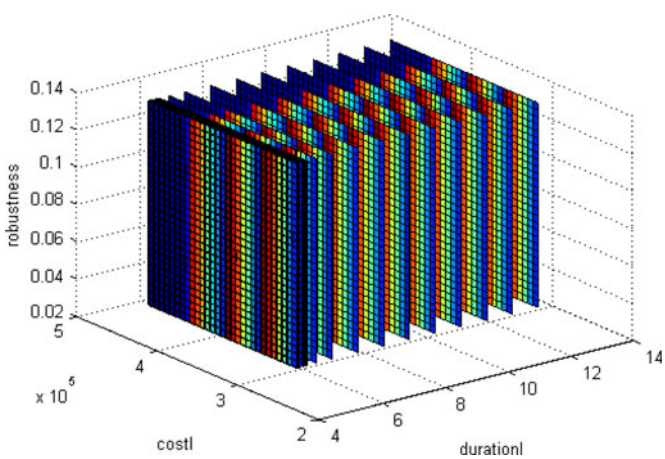


Fig. 10. Slice plot of the 50 percent-summary attainment surface obtained at the scheduling point $t_l = 2.6$ in Real_3.

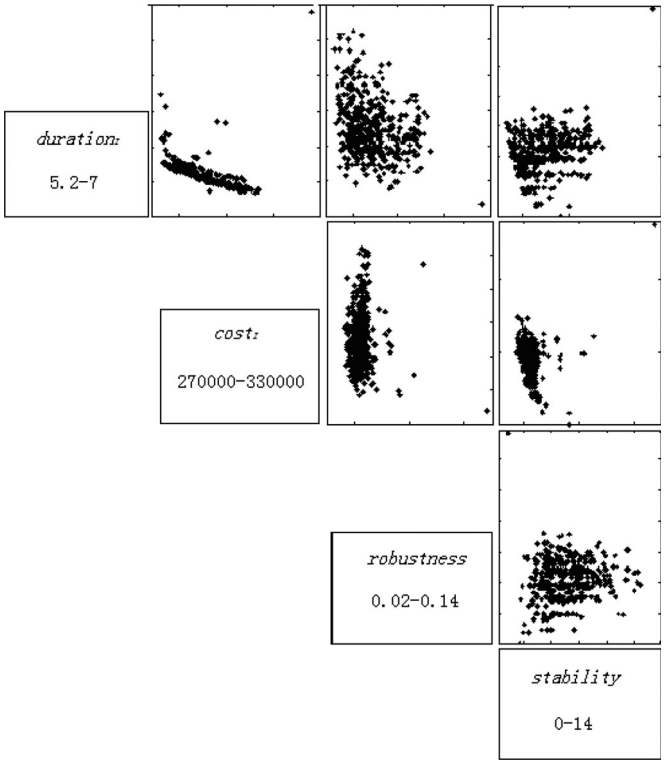


Fig. 11. Diagonal plot of the Pareto front obtained in one run of dε-MOEA at the scheduling point $t_l = 2.6$ on Real_3.

$cost_I$. When finding solutions that have smaller $cost_I$, *robustness* or *stability* becomes worse. However, it is hard to determine the relationship from the figures *robustness* versus *stability*, $duration_I$ versus *robustness* and $duration_I$ versus *stability*. For example, a small *robustness* may correspond to either a small or high *stability*. There is no solution that can simultaneously optimize all the considered objectives.

Table 10 gives several examples of the objective vectors selected from the Pareto front shown in Fig. 11. A solution may perform very well for one objective, but poorly for some others, such as Solution_1-Solution_4. Some solutions may have good (but not the best) values in all objectives, which indicate a good compromise among all the objectives, such as Solution_5-Solution_7. The Pareto front produced by dε-MOEA can provide a software manager with better knowledge about various trade-offs among multiple objectives. It is very helpful for him/her to make an informed decision about the best compromise with regards to his/her preference.

Next, we will suggest the process of how a software manager could make a manual choice based on the Pareto front provided by our approach. The tool implementing our proposed approach displays the plot of different trade-offs among the four objectives as in Fig. 11. The software manager could first pick a given range of cost and durations, if these are the objectives that he/she is most interested in. For example, he/she could decide that he/she is more interested in solutions with higher cost and lower duration. So, he/she would select a few solutions with high cost and low duration in the figure of $duration_I$ versus $cost_I$. Then, he/she could check the different robustnesses and stabilities of these solutions so that a final choice could be made. Alternatively, the software manager could also choose the schedule automatically suggested by our decision making procedure introduced in Section 4.2.4, if he/she wishes to avoid the manual choice.

TABLE 10
Several Examples of Objective Vectors Selected from the Aggregated Pareto Front at the Scheduling Point $t_l = 2.6$ on Real_3

	$[duration_I, cost_I, robustness, stability]$
Solution_1	[5.30, 304597, 0.12, 6.14]
Solution_2	[6.31, 276596, 0.077, 5]
Solution_3	[6.91, 327794, 0.028,13.79]
Solution_4	[5.34, 296656, 0.036, 0]
Solution_5	[5.67, 286808, 0.057, 0.86]
Solution_6	[5.97, 279699, 0.054, 3.86]
Solution_7	[5.75, 283536, 0.061, 2.00]

7 Conclusion

This paper introduced a novel MOEA-based dynamic scheduling method to regenerate new schedules in response to real-time events and uncertainties in MODPSP. Our first contribution is to capture more of the dynamic features of a real-world PSP than previous work, and formulate the problem with one type of uncertainty and three kinds of dynamic events, including: 1) variations of task efforts; 2) new task arrivals; 3) employee leaves; and 4) employee returns.

Our second contribution is the construction of a mathematical model for MODPSP. In this model, considering the updated project state at each scheduling point, four objectives with respect to the project duration, cost, robustness and stability are optimized simultaneously. In addition, three practical constraints, which are the task skill constraints, no overwork constraints, and the maximum headcount constraints, are considered.

Our third contribution is the design of an MOEA-based proactive-rescheduling method to solve MODPSP. A predictive schedule is generated initially using a proactive scheduling method considering task effort uncertainties. During the project, the previous schedule is revised by a rescheduling method dε-MOEA in response to critical dynamic events. dε-MOEA considers the project duration, cost, robustness and stability simultaneously, and employs heuristic initialization strategies, which exploit dynamic event characteristics and history information so that a new schedule is not regenerated from scratch. Furthermore, new methods to handle the task skill constraints, no overwork constraints, and the maximum headcount constraints are proposed.

Our fourth contribution is a comprehensive experimental study of the newly proposed dε-MOEA. The study is based on three groups of comparisons. The first group compared our proactive scheduling method considering the robustness (ε-MOEA-r) with the method without caring about robustness (ε-MOEA-d). Our analyses confirm that ε-MOEA-r reduces the schedule sensitivity to task effort uncertainties significantly with only a small sacrifice in the project duration and cost under the initial scenario. Meanwhile, better robustness can compensate the worse initial duration and cost. The second group compared our rescheduling method dε-MOEA to the heuristic dynamic scheduling which regenerated a new schedule based on a

simple heuristic rule. Our results show that dε-MOEA is very effective in improving the whole project duration and cost, and it is much less sensitive against task effort variances during the dynamic project scheduling process. The third group compared dε-MOEA to state-of-the-art MOEA-based rescheduling methods. Our analyses confirm the benefits that can be obtained by considering robustness and stability together with the project initial efficiency, where project duration and cost deteriorate only slightly when facing task effort uncertainties, and employee assignments and dedications change very little between the new and original schedules, which reduces the potential confusion to both the manager and employees. In addition, these benefits can be produced without severely affecting the initial efficiency. Our results suggest that dε-MOEA outperforms the state-of-the-art dynamic MOEA (dCOEA) for solving MODPSP since it can provide a software manager with a wider range of non-dominated solutions that are much closer to the reference Pareto front.

Although our MODPSP model is an advancement and considers more aspects of reality than previous models, it is still far from capturing all events and factors that can affect project scheduling situations. As indicated in [48] and [49], estimation inaccuracies may be caused by political behaviors, or psychological and economic factors. Our current work assumes that the deviations in effort estimations follow a Gaussian distribution. An empirical validation should be performed to reveal how suitable the Gaussian distribution is to model deviations, and how to best model such deviations. This can be a challenging study that would probably require data collection in terms of deviations in effort estimations obtained during a period of time. After that, our approach could be easily adapted to use such different distributions. In addition, certain factors could also cause the objectives of software scheduling efforts to be affected. As future work, our approach could be modified to deal with changing objectives by considering them as extra dynamic events to be dealt with. Some methods which can involve the participation of the software manager, such as the interview study for collecting information [48], will be used to get feedback from practitioners on how to improve our approach. Besides, more types of uncertainties and dynamic events which may occur during the project execution, such as changes in the task precedence, addition of new employees to the project, and task cancellations should be considered. More characteristics about tasks and employees, such as the employees' experiences, training courses, and the due-date of each task should also be considered, as well as the relationship between such attributes and the performances of MOEAs on MODPSP need to be further studied. Moreover, a thorough empirical validation in industrial contexts should be performed in order to evaluate the practicality of the approach and to further improve it in terms of how close it is to real software development scenarios. In particular, such empirical validation would allow us to get feedback on the assumptions made by our approach, on additional types of uncertainty and dynamic events to be considered, and on the trade-off between the improvements in cost, duration, robustness and stability provided by our approach and the effort needed to adopt the approach in the real world.

Various dynamic events and factors can affect project scheduling situations, thus future PSP investigations should avoid making simplistic modeling assumptions and simplifications that are not valid in practice.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61502239, No. 61329302 and No. 61503191, Natural Science Foundation of Jiangsu Province of China under Grant No. BK20150924 and No. BK20150933, an EPSRC Grant (No. EP/J017515/1) on "DAASE: Dynamic Adaptive Automated Software Engineering", and an EPSRC Grant (No. EP/K001523/1) on "Evolutionary Computation for Dynamic Optimization in Network Environments". This work was done while the first author was with CERCIA, School of Computer Science, University of Birmingham, United Kingdom. The authors are grateful to W. Chen and J. Zhang for providing the data of the three real-world PSP instances. The work of X. Yao was supported by a Royal Society Wolfson Research Merit Award.

References

[1] I. Sommerville, *Software Engineering*, 8th ed. Essex, U.K.: Addison-Wesley, 2006.
[2] L. L. Minku, D. Sudholt, and X. Yao, "Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis," *IEEE Trans. Softw. Eng.*, vol. 40, no. 1, pp. 83–102, Jan. 2014.
[3] E. Alba and J. F. Chicano, "Software project management with gas," *Inf. Sci.*, vol. 177, no. 11, pp. 2380–2401, 2007.
[4] C. K. Chang, M. J. Christensen, and T. Zhang, "Genetic algorithms for project management," *Ann. Softw. Eng.*, vol. 11, pp. 107–139, 2001.
[5] F. Luna, D. González-Álvarez, F. Chicano, and M. A. Vega-Rodríguez, "The software project scheduling problem: A scalability analysis of multi-objective metaheuristics," *Appl. Soft Comput.*, vol. 15, pp. 136–148, 2014.
[6] A. Barreto, M. de O. Barros, and C. Werner, "Staffing a software project: A constraint satisfaction and optimization based approach," *Comput. Oper. Res.*, vol. 35, pp. 3073–3089, 2008.
[7] J. D. Wiest, and F. K. Levy, *A Management Guide to PERT/CPM: with GERT/PDM/CPM and Other Networks*. Englewood Cliffs, NJ, USA: Prentice-Hall. 1977.
[8] D. Golenko-Ginsburg and A. Ganik, "Stochastic network project scheduling with non-consumable limited resources," *Int. J. Production Econ.*, vol. 48, pp. 29–37, 1997.
[9] W. Herroelen, B. D. Reyck, and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments," *Comput. Oper. Res.*, vol. 25, no. 4, pp. 279–302, 1998.
[10] W. N. Chen and J. Zhang, "Ant colony optimization for software project scheduling and staffing with an event-based scheduler," *IEEE Trans. Softw. Eng.*, vol. 39, no. 1, pp. 1–17, Jan. 2013.
[11] F. Chicano, F. Luna, A. J. Nebro, and E. Alba, "Using multiobjective metaheuristics to solve the software project scheduling problem," in *Proc. 13th Annu. Genetic Evol. Comput. Conf.*, 2011, pp. 1915–1922.
[12] C. K. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for software project scheduling with genetic algorithms," *Inf. Softw. Technol.*, vol. 50, pp. 1142–1154, 2008.
[13] M. Di Penta, M. Harman, and G. Antoniol, "The use of search based optimization techniques to schedule and staff software projects: An approach and an empirical study," *Softw.: Practice Experience*, vol. 41, no. 5, pp. 495–519, 2011.
[14] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. New York, NY, USA: McGraw-Hill, 2005.
[15] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[16] C. Le Pape, "Constraint propagation in planning and scheduling," Robot. Lab., Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, CIFE Tech. Rep. TR029, 01/1991, 1991.
[17] W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *Eur. J. Oper. Res.*, vol. 165, no. 2, pp. 289–306, 2005.
[18] S. Gueorguiev, M. Harman, and G. Antoniol, "Software project planning for robustness and completion time in the presence of uncertainty using multi-objective search based software engineering," in *Proc. 11th Annu. Genetic Evol. Comput. Conf.*, 2009, pp. 1673–1680.
[19] J. Xiao, L. J. Osterweil, Q. Wang, and M. Li, "Dynamic resource scheduling in disruption-prone software development environments," in *Proc. 13th Int. Conf. Fundam. Approaches Softw. Eng.*, 2010, pp. 107–122.
[20] C. A. Coello Coello, "Evolutionary multiobjective optimization: A historical view of the field," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 28–36, Feb. 2006.
[21] M. Hapke, A. Jaszkiewicz, and R. Slowinski, "Fuzzy project scheduling system for software development," *Fuzzy Sets Syst.*, vol. 67, no. 1, pp. 101–117, 1994.
[22] S. Lazarova-Molnar and R. Mizouni, "A simulation-based approach to enhancing project schedules by the inclusion of remedial action scenarios," in *Proc. Winter Simul. Conf.*, 2011, pp. 761–772.
[23] (2005). Intaver Institute Inc. Software Project Scheduling under Uncertainties. [Online]. Available: http://www.intaver.com/Articles/Article_SoftwareProjectManagement.pdf
[24] G. Antoniol, M. Di Penta, and M. Harman, "A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty," in *Proc. 10th Int. Symp. Softw. Metrics*, 2004, pp. 172–183.
[25] F. Chicano, A. Cervantes, F. Luna, and G. Recio, "A novel multiobjective formulation of the robust software project scheduling problem," in *Applications of Evolutionary Computation*. New York, NY, USA: Springer, 2012, pp. 497–507.
[26] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.
[27] L. L. Minku and X. Yao, "Software effort estimation as a multiobjective learning problem," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, art no. 35, Oct. 2013.
[28] K. Deb, M. Mohan, and S. Mishra, "Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal," *Evol. Comput.*, vol. 13, no. 4, pp. 501–525, 2005.
[29] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multi-objective optimization," *Evol. Comput.*, vol. 10, no. 3, pp. 263–282, 2002.
[30] X. Shen, and X. Yao, "Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems," *Inf. Sci.*, vol. 298, pp. 198–224, 2015.
[31] J. Fülöp. (2005). Introduction to Decision Making Methods, Working Paper 05-6, Lab. Oper. Res. Decision Syst., Comput. Autom. Inst., Hungarian Acad. Sci., Budapest, Hungary. [Online]. Available: http://academic.evergreen.edu/projects/bdei/documents/decisionmakingmethods.pdf
[32] T. L. Saaty, and L. G. Vargas, "Comparison of eigenvalue, logarithmic least squares and least squares methods in estimating ratios," *Mathematical Modelling*, vol. 5, pp. 309–324, 1984.
[33] M. B. Javanbarg, C. Scawthorn, J. Kiyono, and B. Shahbodaghkhan, "Fuzzy AHP-based multicriteria decision making systems using particle swarm optimization," *Expert Syst. Appl.*, vol. 39, pp. 960–966, 2012.
[34] C. Kim, and R. Langari, "Adaptive analytic hierarchy process-based decision making to enhance vehicle autonomy," *IEEE Trans. Veh. Technol.*, vol. 6, no. 7, pp. 3321–3332, Sep. 2012.
[35] D. P. Bernardon, M. Sperandio, V. J. Garcia, L. N. Canha, A. R. Abaide, and E. F. B. Daza, "AHP decision-making algorithm to allocate remotely controlled switches in distribution networks," *IEEE Trans. Power Del.*, vol. 26, no. 3, pp. 1884–1892, Jul. 2011.
[36] C. Mészáros and T. Rapcsák, "On sensitivity analysis for a class of decision dystems," *Decision Support Syst.*, vol. 16, pp. 231–240, 1996.
[37] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Boston, MA, USA: Addison-Wesley, 1995.
[38] L. A. Maciaszek and B. L. Liong, *Practical Software Engineering—A Case Study Approach*. Essex, U.K.: Addison-Wesley, 2005.
[39] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: Wiley, 2001.
[40] C. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 1, pp. 102–127, Feb. 2009.
[41] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm test suites," in *Proc. ACM Symp. Appl. Comput.*, 1999, pp. 351–357.
[42] E. Zitzler, and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.
[43] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm research: A history and analysis," Dept. Elect. Comput. iEng., Air Force Inst. Technol., Wright-Patterson AFB, OH, USA, Tech. Rep. TR-98-03, 1998.
[44] J. R. Schott, "Fault tolerant design using single and multicriteria genetic algorithm optimization," Master's thesis, Dept. Aeronautics Astronautics, Massachusetts Inst. Technol., Cambridge, MA, USA, 1995.
[45] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[46] J. Knowles, "A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers," in *Proc. 5th Int. Conf. Intell. Syst. Design Appl.*, 2005, pp. 552–557.
[47] R. Shang, L. Jiao, F. Liu, and W. Ma, "A novel immune clonal algorithm for MO problems," *IEEE Trans. Evol. Comput.*, vol. 16, no. 1, pp. 35–50, 2012.
[48] A. Magazinius, S. Borjesson, and R. Feldt, "Investigating intentional distortions in software cost estimation—An exploratory study," *J. Syst. Softw.*, vol. 85, pp. 1770–1781, 2012.
[49] M. Jorgensen, K. H. Teigen, and K. Molokken, "Better sure than safe? Over-confidence in judgement based software development effort prediction intervals," *J. Syst. Softw.*, vol. 70, pp. 79–93, 2004.

Xiaoning Shen received the bachelor's degree in automation and the PhD degree in control science and engineering from the Nanjing University of Science and Technology, Nanjing, P.R. China, in 2003 and 2008, respectively. She is an associate professor at B-DAT & CICAEET, School of Information and Control, Nanjing University of Information Science and Technology. Her main research interests include multiobjective optimization, evolutionary computation, and its applications on software engineering and dynamic production scheduling.

Leandro L. Minku received the PhD degree in computer science from the University of Birmingham, Birmingham, United Kingdom, in 2010. During his PhD, he was the recipient of the Overseas Research Students Award (ORSAS) from the British government. He was also invited to a 6-month internship at Google in 2009/2010. He is currently a lecturer (assistant professor) at the Department of Computer Science, University of Leicester, Leicester, United Kingdom. Prior to that, he was a research fellow at the University of Birmingham. His main research interests include computational intelligence for software engineering, machine learning in nonstationary environments/data stream mining, and ensembles of learning machines. His work has been published in internationally renowned journals such as *IEEE Transactions on Software Engineering*, *ACM Transactions on Software Engineering and Methodology*, *IEEE Transactions on Knowledge and Data Engineering*, and *Neural Networks*. He has been invited to give keynotes and tutorials in his research topics. He is a member of the IEEE.

Rami Bahsoon received the PhD degree in software engineering from University College London for his research on evaluating software architecture stability using real options and he attended London Business School for MBA-level studies in technology strategy and dynamics. He is a senior lecturer in software engineering (associate professor) and leads the software engineering for/in the Cloud Interest group at the University of Birmingham, Birmingham, United Kingdom. The group's research aims at developing architecture and frameworks to support and reason about dependable complex software systems, where the investigations span cloud computing architectures and their economics. He published extensively in the area of economics-driven software engineering, cloud software engineering, and utility computing and coedited a book on *Software Architecture and Software Quality* and another on *Economics-Driven Software Architecture* (published by Elsevier).

Xin Yao is a professor of computer science and the director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications) at the University of Birmingham, Birmingham, United Kingdom. His major research interests include evolutionary computation, ensemble learning, and their applications in software engineering. In particular, he has been working on software effort estimation and software defect prediction using advanced machine learning algorithms and on software project scheduling in dynamic environments. He is a fellow of the IEEE and a distinguished lecturer of the IEEE Computational Intelligence Society (CIS). His received the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 and 2015 *IEEE Transactions on Evolutionary Computation* Outstanding Paper Awards, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 *IEEE Transactions on Neural Networks* Outstanding Paper Award, and many other best paper awards. He received the prestigious Royal Society Wolfson Research Merit Award in 2012 and the IEEE CIS Evolutionary Computation Pioneer Award in 2013. He was the President (2014-2015) of IEEE CIS, and the editor-in-chief (2003-2008) of the *IEEE Transactions on Evolutionary Computation*.