Advances in Engineering Software 98 (2016) 79–96

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Investigating the impact of developer productivity, task interdependence type and communication overhead in a multi-objective optimization approach for software project planning

Constantinos Stylianou[a,*], Andreas S. Andreou[b]

[a] Department of Computer Science, University of Cyprus, 75 Kallipoleos Avenue, PO Box 20537, 1678, Lefkosia, Cyprus
[b] Department of Electrical Engineering/Computer Engineering and Informatics, Cyprus University of Technology, 31 Archbishop Kyprianou Avenue, PO Box 50329, 3036, Lemesos, Cyprus

ARTICLE INFO

Article history:
Received 26 March 2015
Revised 1 February 2016
Accepted 4 April 2016
Available online 19 April 2016

Keywords:
Productivity-based software project planning
Task scheduling
Human resource allocation
Multi-objective optimization
Task interdependence
Communication overhead

ABSTRACT

One of the most important activities in software project planning involves scheduling tasks and assigning them to developers. Project managers must decide who will do what and when in a software project, with the aim of minimizing both its duration and cost. However, project managers often struggle to efficiently allocate developers and schedule tasks in a way that balances these conflicting goals. Furthermore, the different criteria used to select developers could lead to inaccurate estimation of the duration and cost of tasks, resulting in budget overruns, delays, or reduced software quality. This paper proposes an approach that makes use of multi-objective optimization to handle the simultaneous minimization of project cost and duration, taking into account several productivity-related attributes for better estimation of task duration and cost. In particular, we focus on dealing with the non-interchangeable nature of human resources and the different ways in which teams carry out work by considering the relationship between the type of task interdependence and the productivity rate of developers, as well as the communication overhead incurred among developers. The approach is applied to four well-known optimization algorithms, whose performance and scalability are compared using generated software project instances. Additionally, several real-world case studies are explored to help discuss the implications of such approach in the software development industry. The results and observations show positive indications that using a productivity-based multi-objective optimization approach has the potential to provide software project managers with more accurate developer allocation and task scheduling solutions in a more efficient manner.

1. Introduction

The success of a software project relies on delivering end-products on time, within budget and with all the required features and functionality. These goals can be realized only if the necessary planning, organizing, staffing, directing and control activities are carried out correctly by project managers. Furthermore, considering the ever-increasing size and complexity of modern-day software products, development companies face escalating pressure of having to provide software products sooner and cheaper than their competitors in order to remain viable. Therefore, a primary concern for software project managers is to make sure the right developers are selected while effectively balancing the duration and cost of a project, in order to prevent unnecessary cost and schedule overruns in subsequent stages of development and any overall reduction to the level of quality of the end-products. Consequently, it is important for software project managers to have support during resource allocation and scheduling activities through tools that will help them successfully select and arrange the most suitable (teams of) developers to guarantee the success of the software project, whilst satisfying both time and budgetary criteria simultaneously.

Human resource allocation and task scheduling in software development projects is a naturally complex and computationally-intensive process since multiple objectives need to be satisfied. For this reason, the problem is classed as a special case of the resource-constrained project scheduling problem (RCPSP), and therefore is considered to be an NP-hard problem, meaning that there is no algorithm known to be able to solve it in polynomial time [1,2]. Project managers often struggle to use a manual approach because there are many different combinations to be

* Corresponding author: Tel.: +357 25 002099.
E-mail address: cstylianou@cs.ucy.ac.cy (C. Stylianou).

http://dx.doi.org/10.1016/j.advengsoft.2016.04.001
0965-9978/© 2016 Elsevier Ltd. All rights reserved.

examined. Thus, an exhaustive search to find the best solution will often prove inadequate and impractical, especially if they have a limited amount of time at their disposal. As a result, the majority of works view the problem as an operational research problem, where proposed solutions make use of techniques that carry out combinatorial optimization of various software criteria, such as cost, duration, or number of defects. Examples include mathematical modelling methods and computational intelligence techniques, where specialized algorithms are employed to locate optimal or near-optimal feasible solutions as a means of providing better and faster support to decision-makers. Several of these approaches emphasize that the allocation of developers and scheduling of tasks needs to take into account certain attributes of the available workforce, such as the capabilities and experience of developers in certain skills, as well as their cost.

There are, however, several other attributes that should also be addressed, the influence of which could allow project managers in the software industry to make more accurate staffing decisions and estimates during their planning activities earlier on in the project. Specifically, the proposed approach uniquely investigates the inclusion of productivity-related attributes concerning:

- the productivity rate of developers.
- the way in which productivity rates are combined within teams based on the type of work carried out (known as task interdependence type).
- the communication overhead that is incurred when developers work together and collaborate.

Including these attributes during planning activities is important, especially in cases where tasks are allowed to be undertaken by several developers whose efforts need to be combined to produce a task's output. Consequently, these attributes reflect on both the duration and cost of a software project, as well as the quality of the end-product.

The paper contributes to the existing research area in two ways. First, it adapts the RCPSP of human resource allocation and task scheduling in software development projects so as to include these overlooked productivity-related attributes. Second, it presents a novel attempt to solve the problem of minimizing the duration and cost of a software development project in its initial stage with the use of a multi-objective optimization incorporating the productivity-related attributes. In order to help achieve this, several objective and constraint functions are proposed to guide the generation of feasible and optimal solutions. Also, we attempt to include hard, realistic assumptions and constraints concerning the availability and suitability of developers that would normally affect the planning decisions of software project managers. The inclusion of these assumptions adds significantly to the complexity of the adapted RCPSP problem we attempt to solve, which makes it that much harder for the optimization process to find feasible and optimal solutions. We therefore employ several different variations of multi-objective genetic algorithms (MOGAs) to carry out the optimization and pose our first research question:

R1. How do different MOGAs perform in terms of generating (near-)optimal solutions with respect to our proposed approach to resource allocation and task scheduling?

Given the fact that as current technology capabilities are constantly improved, software systems progressively become larger. It is also important for our approach to be applicable for varying sizes of projects undertaken by varying sizes of development companies. Consequently, we examine the issue of scalability by setting our second research question:

R2. How do different MOGAs behave in terms of scalability as the number of tasks and developers increases in our proposed approach to resource allocation and task scheduling?

Furthermore, it is equally important to investigate the implications of the productivity-based attributes in practical software development settings. For this, several real-world projects were investigated. The ultimate goal is to provide an approach that accurately reflects both the manner with which these activities are carried out, and also the factors that may influence decisions taken by software project managers in an automated, efficient and less time-consuming way.

The remainder of the paper is structured as follows: Section 2 provides an overview of recent related attempts that also use optimization techniques to solve the problem of human resource allocation and task scheduling in software projects. Section 3 gives a formal description of our proposed adaption of the RCPSP, which considers task type interdependence, developer productivity and communication overhead. Section 4 presents the multi-objective optimization process adopted to solve the problem. Section 5 explains the experiments carried out to evaluate the proposed approach with regards to the research questions. Section 6 presents the results obtained and the quality indicators employed to help compare the MOGA variations. Section 7 discusses several observations made in real-world projects concerning the applicability of our approach. Section 8 examines potential threats to validity and how certain limitations were addressed. Lastly, Section 9 presents a synopsis with concluding remarks and possible future directions.

2. Related work

The majority of attempts to allocate developers and schedule tasks make use of optimization techniques, where the main goal is to maximize or minimize various software development objectives (most popular being cost and duration) through the use of mathematical modelling methods such as linear programming [3–5], constraint satisfaction [6,7], queuing theory [8,9], and statistical/probabilistic modelling [10], in addition to computational intelligence techniques like genetic algorithms [11,12], swarm intelligence [13–15], and fuzzy logic [16,17], or a combination of these methods [18].

Antoniolet al. [18], Ren et al. [12], and Di Penta et al. [19] attempt to schedule work packages and allocate teams to work packages with the goal of minimizing project duration. These attempts focus on software maintenance projects, where work packages are assigned teams of developers as they occur in time or are postponed until developers with the required expertise are available again. The latter attempt also focuses on minimizing the idle times of developers, that is, the time a developer waits to be reassigned to another task. However, these attempts consider that teams are equally capable of carrying out tasks and require the same amount of time to do so.

Alba and Chicano [20,21] propose an approach to allocate developers and schedule tasks in order to minimize the cost and duration of software projects. The authors employ a genetic algorithm to perform the optimization, in which the duration of tasks is determined by the degree of dedication of each assigned developer as long as the skill requirements of tasks are satisfied. Using the same approach, Xiao et al. [15] made a comparison between genetic algorithms and particle swarm optimization, and their results show that the latter technique yield better solutions. Also, Minku et al. [22,23] attempt to improve the quality of solutions and hit rates of the original approach by normalizing the degree of dedication of developers and incorporating a new penalty for evaluating cost and completion time. The approach, however, considers that all developers with a particular skill will possess it to the same level and, thus, assumes that those developers are interchangeable, which may not be the case in real-world settings.

Kapur et al. [24] and Ngo-The and Ruhe [25] use integer linear programming together with genetic algorithms in order to assign resources and schedule the implementation of features in incremental software development. One of the goals of the optimization is to maximize productivity on the assumption that developers with different levels of skills will naturally have different rates of productivity. This can be used by software development companies as a way to schedule product releases with selected features that lead to an optimum business value. However, one of the assumptions is that only one developer can be assigned to implement a feature, which may not be practical considering that larger software projects may require two or more developers to work together on a task in any phase of development.

In the resource allocation and task scheduling approach proposed by Yanibelli and Amandi [26], the authors take into account the level of effectivity of software developers by assessing the degree to which developers will be effective when assigned to work together on the same task. This information is then used in a genetic algorithm approach that attempts to maximize the effectivity of assigned resources. This approach was later modified, first, with a memetic algorithm [27], and second, with a diversity-adapted simulated annealing method [28] as a way to improve to the quality of the generated solutions. Make-span minimization was also introduced by the authors as an additional criterion to the original approach, hence, transforming it into a multi-objective optimization approach [29]. This was then expanded with the integration of simulated annealing in order to improve the exploitation and exploration search processes of the genetic algorithm [30]. All of these approaches assume that the number of developers required for each task and the level of effectivity between combinations of developers is known in advance, which may pose a problem when two or more developers are assigned together for the first time. Furthermore, the authors completely ignore the cost dimension when allocating and scheduling developers. The most significant downside to this approach, however, is that the duration of each task is not actually influenced by how developers work together. In fact, regardless of how effective the developers assigned to a task are, the duration remains unaffected.

The goal of our approach is to provide support for software project managers in their decision regarding who will work on what and when, giving them the ability to choose from a set of solutions with respect to the cost/time trade-off that best fits the needs of the project and the development company. We propose to allocate developers and schedule tasks taking into account the fact that even though the amount of work required to complete a task is the same regardless of who carries it out, the amount of time it takes for the task to complete depends on the developers assigned to work on it, providing they possess the necessary skills. Existing optimization approaches have tended to overlook this factor, and instead either regard developers as interchangeable (meaning that they all possess the same skills) or focus simply on whether or not developers possess the necessary skills required by a task. In software development projects, unlike in other engineering projects, human resources should not be regarded as interchangeable. Furthermore, in reality, developers possess varying levels of skills depending on the knowledge acquired through education and training, their natural abilities and talents, as well as their experiences accumulated over time. Hence, differences in levels of skill implies differences in productivity rates of developers, which in turn affects the duration and cost of each task, and of the project as a whole. Also, our approach is intended to be used for scheduling tasks and allocating resources at the beginning of a software development project to help project managers make more accurate budget and time estimates. Furthermore, we intentionally allow for more than one developer to be allocated to a task and consider how the contributions of each individual team member are combined with respect to their productivity rate. For this, we adopt Steiner's [31] classification of task interdependence to help select which operator to use for aggregating individual productivity rates depending on the specific type of software development task. Finally, another important factor that also influences the duration and cost of a project, which is often neglected, is the issue of communication overhead. To the best of our knowledge, very few attempts have taken into account the increase in time that could occur when developers work together on a task. One attempt is presented by Di Penta et al. [32], who explore how different models and levels of communication overhead affect the allocation of developers in teams, as well as on the overall make-span of a project in a search-based project staffing and scheduling approach. Our approach adopts a similar rationale by incorporating this factor in the calculation of a task's duration that is then used in the optimization process.

3. Description of adapted RCPSP

A software development project consists of a set $T = \{t_1, t_2, \ldots, t_M\}$ of M tasks, which are determined by the project manager based on the activities from the different phases of software development that will be required. All tasks must be undertaken by at least one software developer employed by the development company. The project manager also determines the set $S = \{s_1, s_2, \ldots, s_K\}$ of K professions whose associated skills will be required by the software project. Each task $t_i \in T$ is subsequently assigned one (and only one) of these professions, denoted by t_i^{prof}, the skills of which are required for the task to be completed. For example, a project manager would assign a task that entails interviewing stakeholders to a requirements analysis profession since he or she will determine that elicitation skills will be necessary to carry out the task.

Next, the software project manager determines which tasks are related to each other in the form of dependency relationships. We assume that only finish-to-start dependency relationships exist, meaning that in order for a task to start, all its predecessor tasks must first finish. The set of dependency relationships D contains pairs of tasks such that $(t_i, t_j) \in D$ if task t_j depends on task t_i. A task precedence graph (TPG) consisting of nodes and edges can be used to help depict the dependency relationship between tasks, where the nodes and edges represent tasks and dependency relationships, respectively. Fig. 1 illustrates an example task precedence graph of a software project consisting of eight tasks and ten dependency relationships.

Once the tasks and dependency relationships have been identified, it is up to the project manager to provide an estimate of the effort or workload that will be required to carry out each task. The effort required for each task $t_i \in T$ is denoted by t_i^{effort}. According to the Project Management Institute [33], effort is defined as "*the number of units of labour needed to complete a scheduled activity or work breakdown structure component*." It is commonly expressed in person-hours (i.e., the number of hours needed for an average developer to carry out the work), though it is possible to represent effort in person-days, person-weeks or even person-months for large projects.

The human resources of a software development company form the set $R = \{r_1, r_2, \ldots, r_N\}$ of N software developers, who are able to participate in a project based on their availability and area of expertise. Each developer $r_i \in R$ is associated with an hourly wage rate (r_i^{salary}), used to calculate the cost of each task to which developer r_i is assigned. The hourly wage rate can be easily obtained from the company's human resource or accounting department.

Often, developers may possess appropriate skills in more than one profession. As a result, they can work on a project in different capacities, thus adding to the complexity of the allocation and

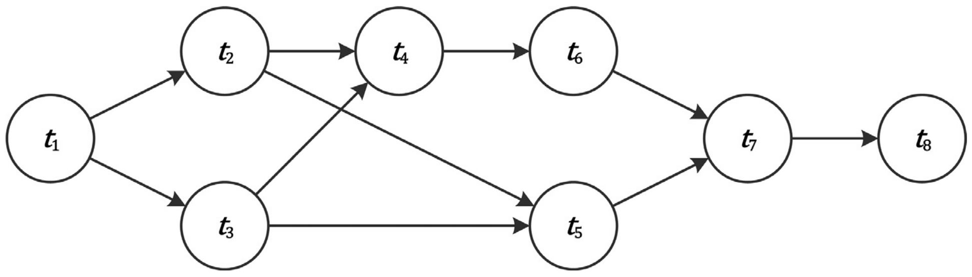

Fig. 1. An example TPG for a software project.

scheduling process. This is particularly true in small-to-medium sized companies, where developers are often forced to undertake several roles in the company due to limited resources. To accommodate this in our approach, each developer $r_i \in R$ is assigned a rate of productivity for each profession $s_j \in S$ required by the software project. This information is represented by a productivity matrix $P = [p_{ij}]$ where p_{ij} denotes the rate at which developer r_i is able to carry out any task belonging to profession s_j. The value of this productivity rate is selected by the project manager from the range [0.0,2.0] and can be determined in any number of ways using the metrics employed by the software development company and then normalized to fit this range. If a development company is mature enough, project managers may be able to adopt principles from standards and frameworks, such as the People CMM [34], which provide ways to quantify productivity using factors such as experience, competency and capacity. Also, project managers may have access to developers' scores in evaluation reports to help determine their productivity rate. Additionally, project managers can use their experience and expertise to assess the technical skills, know-how and performance of developers in past projects with similar tasks to form a productivity profile of their resources. If a developer does not possess any of the skills of a particular profession, then the project manager will assign a rate of productivity equal to 0.0, and the developer will not be considered as a candidate to be assigned to any task belonging to that profession. For an average developer that possesses the skills of a specific profession, the project manager will likely assign a nominal value of 1.0 as a productivity rate for that profession. In the case of a novice or newly-recruited developer that is considered to be half as productive as an average developer in a certain profession, he or she may be assigned a productivity rate of around 0.5 for that profession. On the other hand, for an above-average developer that possesses the skills of a certain profession (such as an expert), a project manager may assign the developer a productivity rate of 2.0 if he or she is considered to be twice as productive as an average developer in that profession.

As mentioned previously, our approach allows for more than one developer to be assigned to a task. Therefore, it is important to identify how developers will work together to produce the output of each task. In order to do this, we adopt the task interdependence categorization proposed in Steiner's taxonomy of group task [31]. Steiner identified different "*combinatorial strategies*" that define the ways with which a team's overall contribution to a task can be measured based on the individual contributions of its members. According to Steiner, there are five different types of task interdependence: additive, compensatory, disjunctive, conjunctive and discretionary, where each type adopts a unique function for aggregating individual productivity in order to determine overall team productivity. The types of task interdependence and their corresponding aggregation functions used in our approach are additive, disjunctive and conjunctive tasks. Hence, the software project manager assigns a task interdependence type (t_i^{type}) to each task $t_i \in T$ taken from set $G = \{$*additive, disjunctive, conjuctive*$\}$.

In the case of additive tasks, the overall productivity of a team is obtained by adding together the individual productivity of its members. Additive tasks are classed as divisible and maximizing, meaning that these tasks can be broken into subtasks, the goals of which focus on the quantity of the output. Examples of additive software development tasks include verification and validation tasks, such as usability inspections and software reviews, where the work can be divided into subtasks in order for developers to attempt to find as many defects as possible [35]. Since developers will work individually in such tasks and then pool together their work, the higher their individual productivity rates, the higher the effectiveness of the team in defect detection. Software development tasks that require brainstorming may also be considered additive tasks. With regards to disjunctive tasks, the overall productivity of a team is equivalent to the highest individual productivity. Disjunctive tasks are unitary and optimizing, indicating that they cannot be further decomposed and that they focus on the output's quality. Database design may be considered an example of a disjunctive task. If a team of developers is assigned to come up with a suitable (optimal) schema, not all developers are required to come up with the best solution. Instead, it is enough for only one member to provide the best solution. Hence, a highly productive member who is able to come up with the best solution quicker will help the team finish such a task sooner compared to a team whose most productive member has an average or lower productivity rate. Similarly, a task involving the integration of two modules can also be regarded as a disjunctive task. For conjunctive tasks, the overall productivity of a team is defined as the lowest individual productivity. Conjunctive tasks can be considered either divisible focusing on quality, or unitary focusing on quantity. Implementation tasks are an example of software development tasks that can be regarded as conjunctive. For example, in the case where the programming/coding of a module has been split into subtasks for team members to implement individually, the developer possessing the lowest rate of productivity out of the whole team will determine the team's overall rate of productivity. Compensatory and discretionary types of tasks are not adopted in our approach as they are not considered to be applicable in the case of software development tasks. In compensatory tasks, the overall productivity of a team can be expressed as the variability of individual productivity. In our approach though, it does not make sense to require developers to have diversity in their rates of productivity. In discretionary tasks, the decision on how to combine individual contribution is left up to the team members. However, in our approach we assume that the output of each software task can only be derived using one specific type.

Finally, each task incurs a communication overhead (t_i^{overhead}) depending on the number of developers assigned to carry out the task. Typically, when developers work together as a group, there is an amount of time spent on communicating with each other in order to coordinate activities, discuss issues and resolve conflicts regarding a task. Communication can take many forms (such as meetings, phone calls, e-mails, video calls, etc.) all of which

Table 1
Correlation between team size, number of communication paths and percentage of communication overhead [38].

Team size	Communication paths	Communication overhead
0	0	0.00%
5	10	1.50%
10	45	6.00%
15	105	13.50%
20	190	24.00%
25	300	37.50%
30	435	54.00%

take away from work that the developers are assigned to perform. We propose, therefore, to take into account this additional time needed for communication by adjusting the make-span of each task in order to give a more accurate project duration and cost.

According to Brooks [36], there is a polynomial relationship linking the size of a team working on a task with the number of possible communication paths between pairs of developers. Specifically, the relationship between the number of developers (t_i^{n}) assigned to task t_i and the number of communication paths (t_i^{paths}) can be defined using Eq. (1).

$$t_i^{\mathrm{paths}} = \frac{t_i^{\mathrm{n}} \times \left(t_i^{\mathrm{n}} - 1\right)}{2} \tag{1}$$

This means that as the number of developers working together on a task increases, so does the number of communication paths, resulting in an exponential growth in communication overhead. Abdel-Hamid and Madnick [37], therefore, carried out an empirical investigation to attempt to quantify the percentage of communication overhead incurred given different team sizes. Their findings are presented in Table 1.

In order to determine the percentage of communication overhead for any specific number of developers, Douglas [38] suggests interpolating between the two nearest team sizes given in Table 1. By applying linear regression, he was then able to formulate an equation (Eq. (2)) using the number of communication paths as a variable to calculate the percentage of communication overhead of any team size. Our approach adopts this formula to calculate the communication overhead (t_i^{overhead}) of each task $t_i \in T$.

$$t_i^{\mathrm{overhead}} = 0.001248269 \times t_i^{\mathrm{paths}} \tag{2}$$

With the above information regarding tasks and developers, the goal is to allocate developers and schedule tasks in such a way that the shortest possible project make-span and cost are achieved simultaneously. It is assumed that a task can be assigned more than one developer, and a developer can be assigned to tasks associated with different professions as long as they possess the required skills and have a positive non-zero productivity rate. Also, a developer can be assigned to work on only one task at any given time. This means that a developer will not be set to work on tasks that are executed concurrently, thus avoiding conflicts in assignment. Furthermore, tasks cannot be pre-empted, that is, once a task starts it must be completed and its execution cannot be suspended. In the case of divisible tasks, a developer may finish his or her contribution earlier than other team members. Normally, this would allow the developer to be free to work on another task. However, in this investigation, we consider that developers are assigned to a task as a team and so all developers will remain assigned until the whole task is complete and they will be paid for the whole duration of the task. In order to achieve these goals and satisfy the underlying constraints, we propose several objective and constraint functions in Section 4 to guide the generation of optimal and feasible solutions.

4. Multi-objective optimization approach

The aim of our approach is to allocate human resources and schedule tasks so that the overall project duration and cost are minimized. We propose to solve our adaptation of the RCPSP by employing a multi-objective genetic algorithm as the optimization method that will generate near-optimal solutions. Genetic algorithms [39] are a type of evolutionary algorithm, which are widely used to solve search-based optimization problems by simulating the theory of natural evolution on a population of individuals (candidate solutions). Each solution is evaluated using an objective function, which assesses how fit a solution is in solving the problem. By iteratively applying mechanisms inspired by the survival of the fittest (such as selection, crossover and mutation) to the solutions, each generation gradually improves the fitness (or quality) of the solutions and discards lower-quality solutions until an optimal solution is located. Often, certain problems require several criteria to be optimized at the same time. To handle this, multi-objective genetic algorithms can be employed, where the goal is to find the best solution by optimizing a vector of objective functions. Sometimes, however, objectives can be conflicting or competing in nature, and in such cases a multi-objective genetic algorithm would generate a set of optimal solutions rather than a single optimal solution [40,41]. This set of solutions is known as the Pareto optimal set and contains only those solutions that are non-dominated by (or non-inferior to) others in the set. In other words, each optimal solution represents a particular trade-off between the objectives, where any improvement in one of the objectives leads to the worsening of one or more other objectives. Because this approach works as an a posteriori method, that is, the decision-maker does not provide any preferences regarding the importance of each objective before the optimization, the decision-maker is free to adopt any one of the optimal solutions generated [42]. In the context of our approach, the individuals represent developer assignments and task schedules that are evolved repeatedly in order to generate a set of non-dominated solutions that minimize the project duration and cost simultaneously based on productivity-related characteristics of the tasks and available developers, as shown in the diagram of our proposed approach (Fig. 2). After examining the generated solutions, a project manager can then decide which one is most suitable to follow based on the trade-off between make-span and cost.

4.1. Representation and encoding

Candidate solutions are represented by individuals in the population that are composed of two variables: one to handle the information regarding the allocation of resources and one to handle the information related to the scheduling of tasks, as similarly proposed by Yannibelli and Amandi [26]. An example of the encoding of a candidate solution using the TPG of Fig. 1 is shown in Fig. 3. The first variable is encoded using a binary array of length M, where each element u of the array contains a sequence of bits that represent only those developers that possess the required skills to carry out task t_u. If the value of a bit in the sequence is “0”, then the corresponding developer is not assigned to the task, whereas if the value of the bit in the sequence is “1”, then the corresponding developer is assigned to the task. With this representation, the total number of bits required always varies according to the number of tasks and number of available developers in each profession. The second variable uses a permutation array whose length is also equal to the number of tasks in the project, M. Each element v of the array contains the index of a task in the project. Tasks can only appear once in the array and are chosen for scheduling in order of their appearance from left to right in the array.

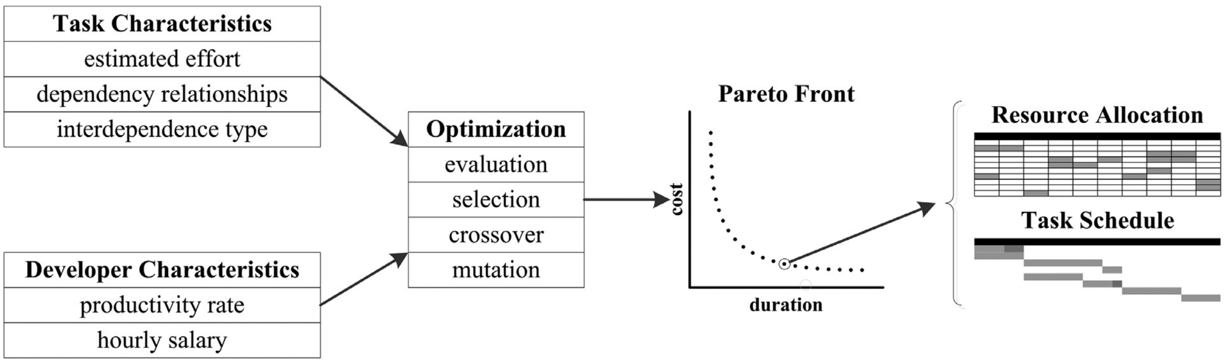

Fig. 2. The elements of the proposed productivity-based multi-objective optimization approach.

u	1	2	3	4	5	6	7	8
	1010	011000	000010	010100	01000010	00111000	10000010	00100

v	1	2	3	4	5	6	7	8
	1	3	2	5	4	6	7	8

Fig. 3. Example of the encoding used to represent an individual: (top) developer allocation variable, and (bottom) task schedule variable.

4.2. Population initialization

Both variables must be initialized subject to several feasibility restrictions in order to guarantee that the candidate solutions they represent are valid. At the same time, the initialization should ensure that the population represents a varied and heterogeneous pool of candidate solutions. This can be achieved by randomly setting the starting values of the two variables subject to the necessary constraints. For the binary array representing the allocation of developers, each task must have at least one developer assigned to it. For the permutation array representing the scheduling order of tasks, each task must appear in the array after all of its predecessors in order for the solution to be feasible. To ensure this, the initialization follows a process of randomly selecting a task only from a set that consists of tasks that have not yet been selected and whose predecessors have already been inserted in the array. Thus, the initial permutation arrays will always represent feasible project schedules, since no task will be selected before its predecessors.

4.3. Solution fitness and feasibility

Our optimization approach adopts two objective functions, $F_{\text{duration}}(x)$ and $F_{\text{cost}}(x)$, in order to assess the fitness of each individual solution x in the population. The values given to these two objective functions denote the duration and the cost of the project, respectively. Also, our approach uses two constraint functions, $G_{assignment}(x)$ and $G_{dependency}(x)$, which assess the feasibility of an individual solution. The first constraint function reflects whether or not at least one developer is assigned to a task, and the second one reflects whether or not any dependency violations between tasks exist. The following subsections describe in detail how the values of each function are calculated.

4.3.1. Project duration objective function

In order to compute the overall duration of a project, $F_{\text{duration}}(x)$, represented by a solution x in the population, *POP*, our approach first calculates the duration of each task individually to determine the amount of time (in hours) that developers will spend on each task they are assigned to. Then, by using the precedence relationships between tasks and the availability of developers, the starting and finishing times of each task are determined. The project duration is then established as the highest finishing time of all tasks.

The duration of a task is calculated using the productivity rate of developers, similarly to the way presented in Kapur et al. [24] and Ngo-The and Ruhe [25], but also using the type of task interdependence. The order in which task durations are computed is determined by the order that tasks appear in permutation array variable. With this forward scheduling approach, the first element of the permutation array (position $v = 1$) contains the index u of the task whose duration is to be calculated first. Using this index, the corresponding element at position u of the binary array variable is decoded in order to obtain the set of developers assigned to the task. Once the set of developers A_i assigned to task t_i has been determined, the duration is then calculated by dividing the effort that t_i is estimated to require by the overall productivity rate (t_i^{prod}) of the team of developers comprising the set A_i, as shown in Eq. (3).

$$t_i^{\text{duration}} = \frac{t_i^{\text{effort}}}{t_i^{\text{prod}}} \tag{3}$$

The overall productivity rate t_i^{prod} is computed using Eq. (4), which takes into account the information in the productivity matrix P concerning each developer $r_k \in A_i$ and the task's interdependence type t_i^{type}.

$$t_i^{\text{prod}} = \begin{cases} \text{sum}\ \{p_{kj} \mid r_k \in A_i\}, & \text{if } t_i^{\text{prof}} = s_j \text{ and } t_i^{\text{type}} = additive \\ \max\ \{p_{kj} \mid r_k \in A_i\}, & \text{if } t_i^{\text{prof}} = s_j \text{ and } t_i^{\text{type}} = disjunctive \\ \min\ \{p_{kj} \mid r_k \in A_i\}, & \text{if } t_i^{\text{prof}} = s_j \text{ and } t_i^{\text{type}} = conjunctive \end{cases} \tag{4}$$

The term p_{kj} denotes the rate of productivity possessed by developer r_k at profession s_j. Using the operationalization suggested by Steiner [31], if task t_i is categorized as additive, then the summation operator is used to aggregate individual productivity rates. Alternatively, if task t_i is categorized as disjunctive, then the maximum operator is applied. Else, if task t_i is categorized as conjunctive, then the minimum operator is.

Algorithm 1 Procedure to compute earliest start and finish time of a task.

```
1. Input: Set of developers $A_i$ assigned to task $t_i$, start time $t_i^{start}$, duration $t_i^{duration}$ and finish time $t_i^{finish}$.
2:
3. conflict ← true
4. while conflict
5.    for all developers $r_k \in A_i$ do
6.       if developer $r_k$ is assigned to any task $t_j$ $(j \neq i)$ at any time unit between $t_i^{start}$ and $t_i^{finish}$
7.          $t_i^{start} = t_i^{start} + 1$
8.          $t_i^{finish} = t_i^{start} + t_i^{duration}$
9.          conflict ← true
10.      else
11.         conflict ← false
12.      end if
13.   end for
14. end while
15.
16. Output: Start time $t_i^{start}$ of task $t_i$ and finish time $t_i^{finish}$ of task $t_i$.
```

With this information, we can then adjust the duration to take into account the communication overhead calculated from Eqs. (1) and (2) accordingly, using Eq. (5) for calculating the duration of a task. The ceiling function rounds up to the nearest hour.

$$t_i^{\text{duration}} = \frac{t_i^{\text{effort}}}{t_i^{\text{prod}}} \times \frac{1}{1 - t_i^{\text{overhead}}} \tag{5}$$

Following this, the starting and finishing times of task t_i can then be calculated. Specifically, the starting time (t_i^{start}) of task t_i is determined by the maximum finishing time out of all its predecessor tasks. If task t_i has no predecessor tasks, then it can begin immediately. Formally, the starting time of a task is expressed using Eq. (6).

$$t_i^{\text{start}} = \begin{cases} 0, & \text{if } \exists t_j \text{ such that } \left(t_j, t_i\right) \in D \\ \max\left\{t_j^{\text{finish}} \mid \left(t_j, t_i\right) \in D\right\}, & \text{otherwise} \end{cases} \tag{6}$$

The finishing time (t_i^{finish}) of a task t_i is simply equal to the starting time plus its duration, as shown in Eq. (7).

$$t_i^{\text{finish}} = t_i^{\text{start}} + t_i^{\text{duration}} \tag{7}$$

As previously mentioned, the durations of the tasks are computed using the order in which they appear in the task schedule variable. Hence, each task is placed at the earliest possible starting time taking into account the finishing times of its predecessor tasks. However, before doing this, it is necessary to examine the availability of the assigned developers in order to avoid any conflicts that will cause a schedule to be infeasible. The steps carried out to handle this are presented in Algorithm 1. First, the start and finish times of task t_i are calculated using Eqs (6) and (7), respectively. If all developers $r_k \in A_i$ are available for the duration of task t_i, then no modifications to the start and finish times are necessary. However, if at least one of the developers assigned to carry out task t_i is already assigned to a different task between the start time and finish time of task t_i, then the value of t_i^{start} is adjusted to the next time step and t_i^{finish} is recalculated again using Eq. (7). This process repeats until the earliest time is determined that satisfies that all developers are available to work for the whole duration of the task.

Once all task start and finish times have been determined, then the overall duration of the software project represented by solution x is calculated by taking the value corresponding to the highest finishing time out of all M tasks. This value corresponds to the value given to the first objective function, $F_{\text{duration}}(x)$, and is defined in Eq. (8).

$$F_{\text{duration}}(x) = \max\left\{t_i^{\text{finish}} \mid t_i \in T\right\} \tag{8}$$

4.3.2. Project cost objective

To calculated the overall cost of a software project, $F_{\text{cost}}(x)$, represented by a solution x in the population, *POP*, our approach begins by calculating how much the assigned developers will cost for each task, and then aggregating all individual task cost. Specifically, the cost (t_i^{cost}) of task t_i is computed by Eq. (9), which aggregates how much each assigned developer $r_k \in A_i$ will cost for the duration of the task based on his or her wage rate.

$$t_i^{\text{cost}} = \sum_{k=1,\ r_k \in A_i}^{t_i^{\text{n}}} \left(t_i^{\text{duration}} \times r_k^{\text{salary}}\right) \tag{9}$$

Subsequently, the overall cost of developers for the project represented by solution x is computed by summing the cost of all M tasks individually. This value corresponds to the value given to the second objective function, $F_{\text{cost}}(x)$, and is defined in Eq. (10).

$$F_{\text{cost}}(x) = \sum_{i=1}^{M} t_i^{\text{cost}} \tag{10}$$

4.3.3. Assignment and dependency constraints

The feasibility of each candidate solution x in the population is assessed by again using the information stored in the two arrays. Since our approach uses forward scheduling based on the availability of the assigned developers, there is never a violation that a developer is assigned to more than one task at any given time. Hence, the only constraints evaluated concern (a) whether or not a task is assigned at least one developer and (b) whether the scheduling of tasks conforms to the precedence relationships. For the former, the value is equivalent to the number of tasks that have no developers assigned, which is calculated by the conditional summation in the constraint function of Eq. (11). For the latter, the value equals to the number of dependencies violated by the schedule, which computed by the conditional summation given in the constraint function of Eq. (12).

$$G_{\text{assignment}} = \sum_{i=1}^{M} \left[t_i^{\text{n}} = 0\right] \tag{11}$$

$$G_{\text{dependency}} = \sum_{(t_i, t_j) \in D}^{|D|} \left[t_j^{\text{start}} \leq t_i^{\text{finish}}\right] \tag{12}$$

4.3.4. Optimization function

The goal of the optimization is to consider the productivity-related characteristics of tasks and developers so as to perform allocation and scheduling in a way that simultaneously minimizes project duration and project cost of a solution x of the population, *POP*, bound by the constraints. At the same time, the two objectives are competing in nature, that is, attempting to decrease one objective would lead to an increase in the other. The optimization will therefore provide a set of optimal solutions rather than a single optimal solution. Thus, the goal is to minimize a vector consisting of the two objective functions of Eqs (8) and (9) depending on the constraints of Eqs (11) and (12), as shown in Eq. (13).

$$\text{Minimize } F(x) = (F_{\text{duration}}(x), F_{\text{cost}}(x)) \text{ subject to } G_{assignment}(x) = 0 \text{ and } G_{dependency}(x) = 0,\ x \in POP \tag{13}$$

For each individual x in the population, *POP*, the two objective functions are evaluated simultaneously. As the algorithm attempts to improve the quality of the population, during selection individuals are compared against each other using both their objective function values to ascertain which individuals are non-dominated as previously described in the beginning of Section 4. Ultimately, since each solution represents a different allocation and scheduling plan, the project manager will be offered a choice on which plan he or she feels suits the project and the organization better.

4.5. Genetic operators

In our approach, two binary tournaments are performed in order to select the parents to create offspring. In each tournament, a pair of individuals is randomly selected as candidate parents and then compared with each other based on their fitness (i.e., objective function values). The individual with the highest fitness is declared "winner" of the tournament and is chosen as a parent. In this way, individuals with greater fitness have a better chance of becoming parents and surviving into the next generation. If the fitness of two candidates is tied, then the candidate with the lowest number of violations as determined by the constraint functions is chosen. For the crossover operator, the developer allocation variable uses single-point crossover at a random bit of the binary array, whereas the task scheduling variable uses partially-mapped crossover [43], which guarantees that the constraint of having each task only appear once is satisfied. For the mutation operator, the developer allocation variable uses a bit-flip operator where a randomly selected bit is altered from a value of "0" to a value of "1" or vice-versa. For the task scheduling variable, a swap mutation takes place where two positions in the permutation array are chosen randomly and the indices in those positions are swapped. In this way, the preservation of the validity of an individual is ensured.

5. Experimental process

5.1. RQ1: Performance of multi-objective genetic algorithms

The first research question investigates whether certain variations of MOGA are able to perform better with respect to locating (near-)optimal developer allocations and task schedules, in view of the increased optimization complexity that results from the addition of realistic assumptions and constraints in our productivity-related adaptation of the RCPSP. Therefore, to answer the first research question, we selected to compare four well-known variations of MOGAs, namely, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [44], the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [45], the Pareto Archived Evolution Strategy (PAES) [46], and the Multi-Objective Cellular algorithm (MOCell) [47]. We selected these algorithms because they have been extensively used as the underlying mechanism to solve related project scheduling optimization problems in the past. In addition, their use would allow similar future research attempts to compare with our proposed approach.

A dataset (DS1) containing sixteen synthetic software projects of varying size, both in terms of the total number of tasks involved, as well as the total number of available developers was then generated, subject to several conditions. To begin with, four distinct software development projects were constructed with varying size of *M*: (i) 25 tasks, (ii) 50 tasks, (iii) 75 tasks, and (iv) 100 tasks. Each task in a project was randomly assigned to one of nine software development professions shown in Table 2, which were identified using the Standard Occupational Classification System [48] and the O∗NET Resource Center [49].

Table 2
List of software development professions used in creation of synthetic software projects.

1. Computer and information systems managers
2. Information security analysts
3. Computer systems analysts
4. Computer systems engineers/architects
5. Database architects
6. Computer network architects
7. Computer programmers
8. Web developers
9. SQA engineers and testers

In addition, the task precedence graph of each project was created in such a way so as no circular dependencies existed among tasks. Also, each task was assigned a task interdependence type (additive, conjunctive or disjunctive). For the sake of experimentation, this assignment was generated randomly, though in further investigations development tasks are assigned specific task interdependence types. Finally, the tasks in each project were randomly given an estimated effort value. Next, four separate sets of software developers were randomly generated with different sizes for *N*: (a) 25 developers, (b) 50 developers, (c) 75 developers, and (d) 100 developers. A productivity matrix was then randomly generated containing values in the range of [0.0, 2.0] making sure that each profession had at least two developers with a value greater than zero to guarantee that all tasks will be able to be completed. Additionally, each developer was assigned a salary indicating his or her wage rate per hour. Salaries were randomly generated within scale ranges in order to reflect that developers with higher levels of expertise and proficiency in skills are more likely to possess a higher productivity rate and, subsequently, cost more in a development company. Finally, each of the four software projects (i)-(iv) was paired with each of the four sets of available workforce (a)-(d) to form 16 project instances.

5.2. RQ2: Scalability of multi-objective genetic algorithms

The second research question assesses the behaviour of the MOGA variations with respect to scalability as the number of tasks and number of available developers increase. To answer this question, we made use of the project instances provided by Luna et al. [50,51], which were intended for use in experiments that adopted the approach presented by Alba and Chicano [20,21]. This dataset (DS2) contains randomly generated projects consisting of six different task sizes (16, 32, 64, 128, 256, and 512) each paired with six different sizes of available developers (8, 16, 32, 64, 128 and 256) for a total of 36 project instances. Because of the underlying differences between approaches, several data present in the instances were then adapted to meet the data requirements of our proposed approach. For example, the instances contained data regarding the skills possessed by developers. This had to be transformed into developer professions so that a developer productivity matrix could be randomly generated for each project instance. In addition, task interdependence types were not present in the instances. Therefore, these also had to be randomly generated based on the number of tasks and number of available developers. The effort required for each task, task precedencies and salary of

Table 3
Parameters and algorithm settings used in execution of the MOGA variations.

Parameter	Value	
Population size	100	
Selection operator	Binary tournament	
Crossover probability	0.90 (single-point)	0.90 (partially-mapped)
Mutation probability	$1/L$ (bit-flip mutation)	0.90 (swap mutation)
Stopping condition	500,000 objective function evaluations	
Number of runs per algorithm	100	

developers in all project instances were left as provided in the original dataset.

5.3. Execution

The representation scheme, objective functions and constraint functions explained in Section 4 were implemented for all four MOGA variations using jMetal 4.3 – a Java-based framework for multi-objective optimization [52]. The same parameters and algorithm settings were used for all algorithms throughout all instances, as summarized in Table 3. Preliminary runs for 50,000 and 100,000 fitness evaluations were performed in order to investigate the convergence of the algorithms with respect to the quality of solutions. Even though the results obtained in these runs showed that the number of fitness evaluations did not actually influence which of the four algorithms performed better, they did show that the quality of solutions could be improved by increasing the number of fitness evaluations. Therefore, in order to allow for a satisfactory trade-off between convergence and computational time, each of the algorithms was executed for 500,000 fitness evaluations.

6. Results and discussion

For each project instance, the four algorithms were run 100 times resulting in the generation of 100 Pareto fronts, each consisting of a number of non-dominated solutions in the objective space corresponding to developer allocations and task schedules. Subsequently, by combining the 100 Pareto fronts, we were able to extract an approximation Pareto front containing the best solutions each algorithm managed to locate for a project instance over its 100 runs. Then, by combining the approximation Pareto fronts of all four algorithms, we were able to construct a reference Pareto front consisting of the overall best solutions found for each project instance. Consequently, each project instance had four approximation Pareto fronts (one for each algorithm) and one reference Pareto front (combining the best solutions of all algorithms). Fig. 4 displays the approximation and reference Pareto fronts achieved for dataset DS1.

We can see that in the smaller-sized project instances, the approximation Pareto fronts of individual algorithms overlap the reference Pareto front, indicating possibly that they are as equally able to find the same near-optimal solutions. However, as the size of the projects increases, both in terms of tasks and developers, we observe that fewer overlaps with the reference Pareto front occur, as well as greater differences in the shape of the individual approximation curves. This could mean that each algorithm is able to locate near-optimal solutions in different regions of the solution space.

6.1. Quality indicators

The hypervolume [53] and inverted generational distance [42] quality indicators were selected to help compare the four algorithms with respect to performance and scalability given their ability to assess both convergence and diversity (uniformity and spread) of algorithms. Specifically, the hypervolume (HV) indicator assesses the volume covered by the non-dominated solutions of a Pareto front in the objective space. Therefore, the larger the volume covered by the solutions generated in a run, the higher the HV value, which indicates a better performance. Since each algorithm was run 100 times, 100 corresponding HV indicator values were calculated for each algorithm for each project instance. The inverted generational distance (IGD) indicator assesses how far the elements of the true Pareto front are from the non-dominated points of an approximation Pareto front. Therefore, the greater the extent of the true Pareto front that is covered by the non-dominated points generated by a run in the objective space, the lower the IGD value, which denotes a better performance. In our case, because it is not possible to know the true Pareto front a priori, the reference Pareto front is used instead. Similarly, 100 IGD values were calculated for the 100 runs of each algorithm for each project instance.

6.2. RQ1: Comparison of performance

In order to compare the performance of the four algorithms, the median HV and IGD values for each algorithm were calculated for each project instance in dataset DS1. The values are presented in Tables 4 and 5, respectively. The shaded cells indicate which algorithm(s) achieved the best value in each project instance (highest median value in the case of the HV indicator or the lowest median value in the case of the IGD indicator). In addition, the average rank of each algorithm is also given.

With respect to the HV indicator, it can be seen from Table 4 that the approximation Pareto fronts generated by MOCell managed to achieve the best median value in ten of the sixteen instances, while the approximation Pareto fronts reached by NSGA-II and SPEA2 achieved the best (highest) median value five and three times, respectively. PAES was the only algorithm not to achieve the highest median value in any of the instances. It should be noted that when computing HV values for a project instance, the solutions of each algorithm are normalized using the reference Pareto front extracted for that particular project instance. As a result, any non-dominated solutions on an approximation Pareto front that are outside of the limits of the reference Pareto front are disregarded. For this reason, a median HV value of zero for certain algorithms can be observed in some project instances [50,51]. Regarding the IGD quality indicator, Table 5 shows that MOCell is once again the predominant algorithm; it attains the best (lowest) median value in eleven of the sixteen instances. NSGA-II achieves the best median value in five instances again, while SPEA2 achieves the best median value six times. The approximation Pareto fronts generated by PAES never obtained the lowest median value for the IGD indicator in any of the instances.

In order to assess which algorithm(s) perform better for our approach, we first carried out a Friedman rank sum test on both quality indicators (with level of significance $\alpha = 0.05$) to detect whether or not a statistically significant difference exists among the four algorithms. For the HV indicator, the test produced a

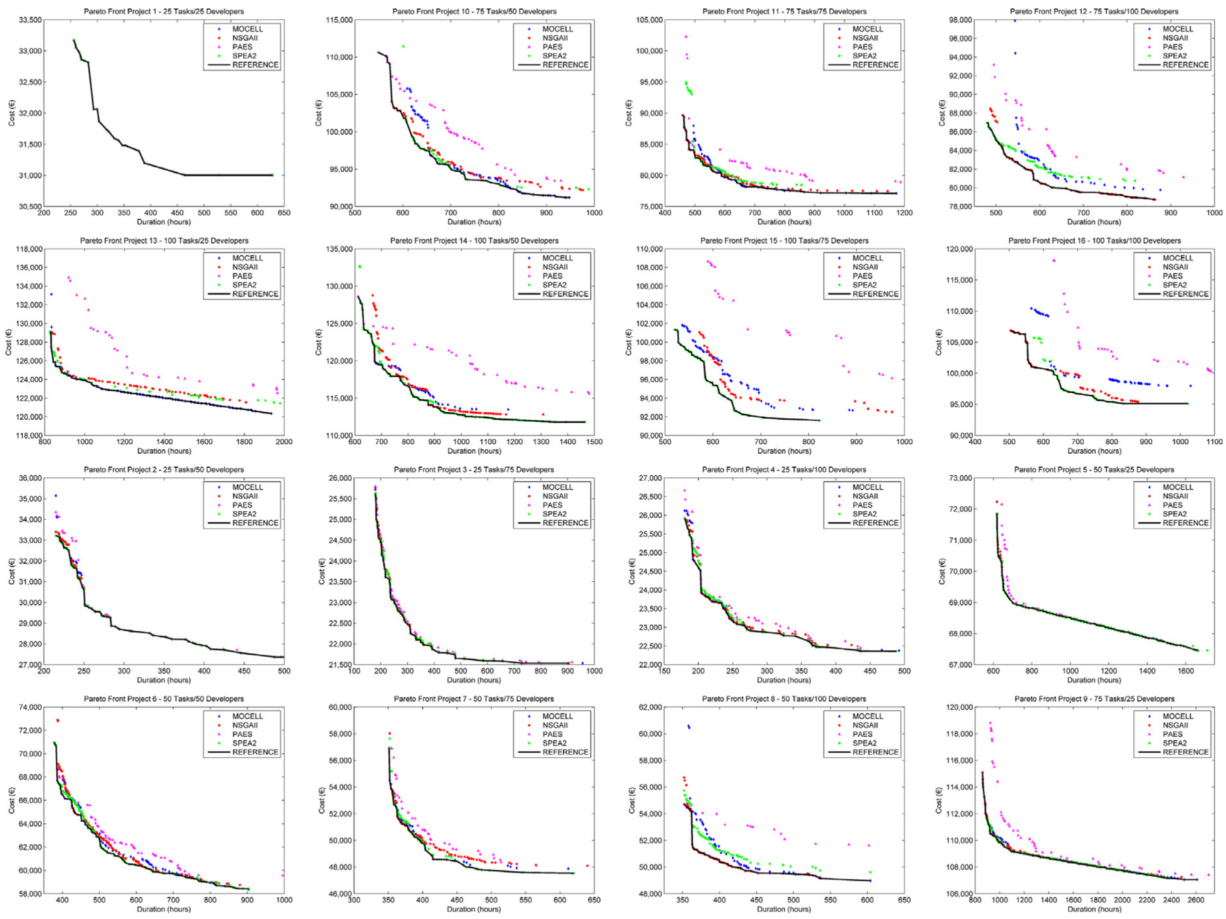

Fig. 4. Reference and approximation Pareto fronts for all algorithms for the sixteen project instances in DS1.

Table 4
Median HV values obtained after 100 runs of each algorithm for project instances in DS1.

Project instance			MOGA variation			
			MOCELL	NSGA-II	PAES	SPEA2
1	25 Tasks	25 Developers	0.9231	0.9231	0.9106	0.9231
2	25 Tasks	50 Developers	0.9533	0.9486	0.9333	0.9431
3	25 Tasks	75 Developers	0.8028	0.7908	0.7250	0.7585
4	25 Tasks	100 Developers	0.8333	0.8438	0.6501	0.7867
5	50 Tasks	25 Developers	0.7364	0.7484	0.6286	0.7468
6	50 Tasks	50 Developers	0.6206	0.6120	0.4925	0.5886
7	50 Tasks	75 Developers	0.6551	0.6733	0.4891	0.6189
8	50 Tasks	100 Developers	0.1773	0.1217	0.0000	0.0915
9	75 Tasks	25 Developers	0.7942	0.7771	0.6175	0.7687
10	75 Tasks	50 Developers	0.3755	0.3848	0.1940	0.3916
11	75 Tasks	75 Developers	0.4295	0.4280	0.1431	0.3883
12	75 Tasks	100 Developers	0.3951	0.3911	0.1648	0.2575
13	100 Tasks	25 Developers	0.5352	0.4754	0.1894	0.4638
14	100 Tasks	50 Developers	0.4765	0.4761	0.1391	0.4469
15	100 Tasks	75 Developers	0.0841	0.0842	0.0000	0.0701
16	100 Tasks	100 Developers	0.0935	0.0986	0.0000	0.1388
		Average ranking	1.6250 (1)	1.7500 (2)	4.0000 (4)	2.6250 (3)

Friedman statistic $\chi_F^2 = 35.3846$, p-value: 1.010×10^{-07}, whereas for the IGD indicator the test returned a Friedman statistic $\chi_F^2 = 34.6212$, p-value: 1.464×10^{-07}. For both indicators, the critical chi-square value at $\alpha = 0.05$ for $k - 1 = 3$ degrees of freedom is computed at 7.815, which is lower than the respective statistics. Hence the tests led to the rejection of the null hypothesis that the algorithms are equivalent with respect to both the HV and IGD indicators. Since the Freidman tests strongly suggested that significant differences do exist between at least two algorithms, a multiple pairwise comparison of algorithms was carried out to identify exact differences between pairs of algorithms. To handle the family-wise error rate accumulated, p-values were adjusted using a post-hoc Holm procedure. The results of the comparison are shown in Table 6, where pairs of algorithms with a statistically significant difference ($p < 0.05$) are shown shaded.

According to the pairwise comparisons, no significant difference is observed between MOCell and NSGA-II, MOCell and SPEA2, and NSGA-II and SPEA2 in either indicator. However, MOCell, NSGA-II

Table 5
Median IGD values attained after 100 runs of each algorithm for project instances in DS1.

Project instance			MOGA variation			
			MOCELL	NSGA-II	PAES	SPEA2
1	25 Tasks	25 Developers	0.0001	0.0001	0.0007	0.0001
2	25 Tasks	50 Developers	0.0014	0.0014	0.0015	0.0014
3	25 Tasks	75 Developers	0.0066	0.0073	0.0111	0.0086
4	25 Tasks	100 Developers	0.0144	0.0153	0.0185	0.0154
5	50 Tasks	25 Developers	0.0009	0.0009	0.0021	0.0009
6	50 Tasks	50 Developers	0.0128	0.0127	0.0170	0.0144
7	50 Tasks	75 Developers	0.0160	0.0150	0.0272	0.0177
8	50 Tasks	100 Developers	0.0536	0.0597	0.0859	0.0638
9	75 Tasks	25 Developers	0.0081	0.0082	0.0124	0.0080
10	75 Tasks	50 Developers	0.0280	0.0272	0.0418	0.0265
11	75 Tasks	75 Developers	0.0301	0.0304	0.0572	0.0330
12	75 Tasks	100 Developers	0.0326	0.0327	0.0498	0.0412
13	100 Tasks	25 Developers	0.0189	0.0220	0.0435	0.0225
14	100 Tasks	50 Developers	0.0215	0.0216	0.0432	0.0229
15	100 Tasks	75 Developers	0.0398	0.0399	0.0951	0.0413
16	100 Tasks	100 Developers	0.0385	0.0382	0.0666	0.0345
		Average ranking	1.6250 (1)	1.9375 (2)	4.0000 (3)	2.4375 (4)

Table 6
Adjusted p-values resulting from the pairwise comparison ($\alpha = 0.05$) for hypervolume and inverted generational distance indicators.

HV	NSGA-II	PAES	SPEA2
MOCELL	0.784191	0.000001	0.085379
NSGA-II	–	0.000004	0.110468
PAES	–	–	0.010365
IGD	**NSGA-II**	**PAES**	**SPEA2**
MOCELL	0.546643	0.000001	0.225180
NSGA-II	–	0.000031	0.546643
PAES	–	–	0.002475

Table 7
Average median HV values per algorithm for instances with the same number of tasks over all sizes of available developers.

Task size	MOGA variation			
	MOCELL	NSGA-II	PAES	SPEA2
16	0.4130	0.4190	0.3859	0.4099
32	0.2692	0.2765	0.2262	0.2687
64	0.0846	0.0805	0.0000	0.0763
128	0.0681	0.0946	0.1384	0.0893
256	0.0037	0.0000	0.1799	0.0000
512	0.0000	0.0000	0.3151	0.0000

and SPEA2 all have statistically significant differences with PAES. Since the HV and IGD indicators relate to the convergence and diversity of a Pareto front, the approximation Pareto fronts generated by MOCell, NSGA-II and SPEA2 can be considered to cover a larger volume of the objective space and are nearer to the optimal compared to PAES. Therefore, based on the statistical analysis these three algorithms would perform better as the underlying multi-objective optimization mechanism for our productivity-based approach, since they are equally capable of generating a more diverse range of trade-offs between project duration and project cost that correspond to developer assignments and task schedules.

6.3. RQ2: Comparison of scalability

In order to compare the scalability of the algorithms, we examined our approach with the 36 instances found in dataset DS2, and followed the method described in Luna et al. [51]. Scalability was assessed in terms of number of tasks and available developers separately, again using the HV indicator as a basis of comparison due to the fact that this metric considers the diversity of solutions and also the convergence of algorithms. In the same way as described previously, the median HV value over 100 runs of each algorithm was calculated for all thirty-six instances.

First, we assessed how the algorithms behave as the size of the projects increases in terms of number of tasks. To do this, for each algorithm we grouped project instances having the same number of tasks and then averaged the median HV values. Table 7 shows these averages for all six different task sizes in dataset DS2. From the table, for example, we can see that for NSGA-II, the averaged HV value of the six project instances with 64 tasks is 0.0805. The results of the table are also shown graphically in the left bar chart of Fig. 5. It is generally expected that the higher the number of tasks, the harder it will be for the algorithms to find near-optimal solutions. Indeed, this does hold true in our case, where it is observed that as the number of tasks increases from 16 to 64, the averaged HV values tend to worsen for all algorithms with a steep slope. For MOCELL, NSGA-II and SPEA2, the increase between 64 and 128 tasks shows a steady behaviour of the algorithms with respect to scalability as there is little change in the averaged HV values. Interestingly, as the number of tasks increase from 128 to 512, the averaged HV value actually increases for PAES.

In a similar fashion, we evaluated how the algorithms behave as the size of the projects increases in terms of number of available developers. For each algorithm, again we grouped instances of projects, but this time by those having the same number of developers, and then averaged their HV values. The averages for all six developer sizes of dataset DS2 are presented in Table 8 and the equivalent bar graph is shown in the right bar chart Fig. 5. For example, we can see that, according to the table, the averaged HV value of the six project instances with 32 available developers for SPEA2 is 0.2687. Once more, it is generally expected that as the number of developers increase, the HV values will decrease. This is mirrored in the behaviour of the MOCELL, NSGA-II and SPEA2 algorithms, which show that they do not scale considerably well, but rather have a sharp gradient. On the other hand, the averaged HV values for PAES show that the algorithm exhibits a better ability for scaling. Overall, the results indicate that PAES is superior with respect to scalability of both task size and developer size, which is similarly concluded in Luna et al. [50,51].

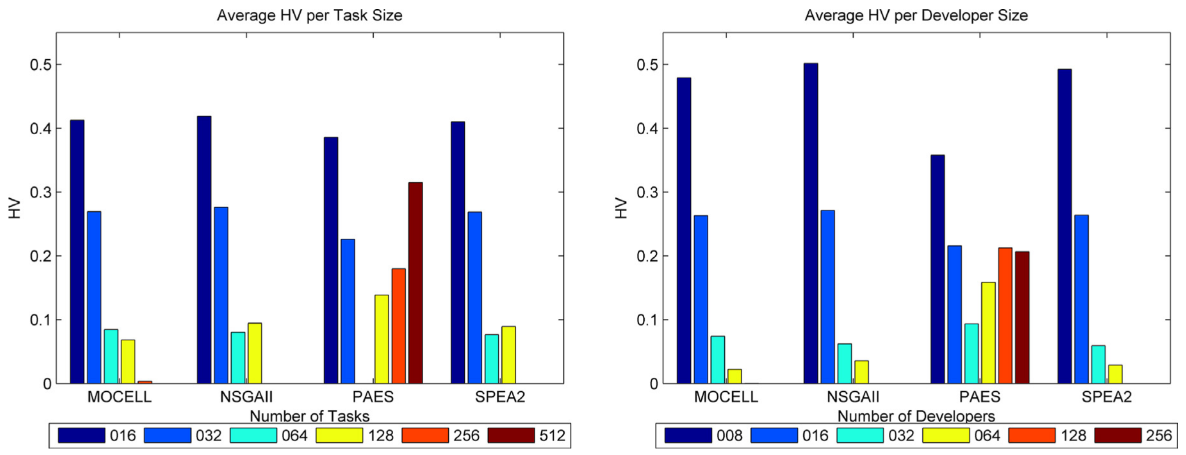

Fig. 5. Comparison of average HV values per algorithm for instances with the same number of: (left) tasks over all sizes of developers, and (right) developers over all sizes of tasks.

Table 8
Average median HV values per algorithm for instances with the same number of available developers over all sizes of tasks.

Developer size	MOGA variation			
	MOCELL	NSGA-II	PAES	SPEA2
8	0.4793	0.5017	0.3582	0.4924
16	0.2631	0.2714	0.2159	0.2636
32	0.0739	0.0619	0.0938	0.0594
64	0.0222	0.0356	0.1584	0.0289
128	0.0001	0.0000	0.2126	0.0000
256	0.0000	0.0000	0.2066	0.0000

7. Real-World observations and considerations

As part of our research, we also examined several software projects that were carried out by third-year university students during a project-oriented Software Engineering course spanning two semesters. The students possessed software development knowledge and skills at different levels of proficiency, and also had gained various practical experiences from their university coursework and, in some cases, industry employment. Groups of four to five students were required to develop a software product following a traditional waterfall life-cycle model and plan-driven development based on requirements from external real-world clients. In some cases, group members worked on tasks individually, whereas in other cases, tasks required two or more group members to work together as a team. Considering that the projects were approximately of the same size, difficulty and complexity, we were able to make several comparisons regarding how the characteristics of the tasks and the productivity rates of the students influenced the duration of tasks. First, we noticed that in tasks with a conjunctive interdependence, such as various programming tasks, if at least one student in a team had a relatively low productivity rate then the whole team would struggle and take longer than planned to complete such a task. We also observed that in tasks with a disjunctive interdependence, such as database designing, even if the majority of the members were not familiar with the task's content, as long as there was one member who possessed the necessary skills and had a high productivity rate, that member was able to help the whole team finish the task sooner. In addition, we noted that in additive tasks, such as the execution of test-cases in the testing phase, students working together with more knowledge in testing had a higher combined productivity rate and, as a result, took shorter time to complete these types of tasks in contrast to teams whose overall team productivity rate was lower. These observations help validate the applicability of our approach that indeed developers combine their efforts in different ways depending on the type of work that needs to be performed and their rate of productivity, which subsequently affects task completion times.

To further investigate the proposed approach, we conducted an additional experiment using a real-world software project undertaken by a local IT company concerning the development of a vessel policies management system for a large insurance brokers company. The supervising project manager at the time of the project had just over five years of industry experience in software project management, and was responsible for the initial planning at the beginning of the project, aiming to find a balance between the total duration and cost of the project. Table 9 presents the characteristics of the project, which comprised 31 tasks split into a number of software development activities (professions). The table also shows the type interdependence of each task, which the project manager helped define according to the nature of the activities in the project. The human resources available to undertake the project (presented in Table 10) consisted of four developers with skills and expertise in one or more of the professions required by the project tasks. The project manager was consulted to provide the estimated effort for each task, as well as the productivity matrix and salary of the developers. Finally, he also provided the Gantt chart he constructed for the project (Fig. 6).

Using the same parameters and settings as before, we conducted 100 runs for all four algorithms and subsequently computed the HV quality indicator from the solutions generated. The results obtained followed to a large degree the same pattern that was observed with the previous experimental software projects. Specifically, NSGA-II and MOCell both managed to outperform PAES and SPEA2 with respect to the quality indicator, demonstrating their ability to generate solutions with higher diversity and cover the extent of the reference Pareto front to a larger degree. Furthermore, there was no statistically significant difference between the results obtained for NSGA-II and MOCell, which suggests once more that these two optimizers are equally suitable for our approach. The reference Pareto containing the overall best solutions is displayed in Fig. 7. Also, in the same figure, we plot the duration and cost of the project corresponding to the original allocation of resources and schedule of tasks constructed by the project manager.

The 27 solutions on the reference Pareto front all dominate the initial estimate made by the project manager either in terms of duration or in terms of cost. The solutions correspond to project plans ranging from short make-spans at higher costs to low costs with longer make-span. The project manager's goal was to

Table 9
Task characteristics of real-world software project.

Task	Effort	Type	Profession	Task	Effort	Type	Profession
T1	48	Disjunctive	Req. Analysis	T16	8	Disjunctive	Testing
T2	16	Conjunctive	GUI Design	T17	12	Additive	Programming
T3	8	Conjunctive	DB Design	T18	4	Conjunctive	DB Design
T4	8	Conjunctive	DB Design	T19	6	Conjunctive	Programming
T5	6	Additive	Req. Analysis	T20	6	Conjunctive	Programming
T6	8	Disjunctive	Testing	T21	64	Conjunctive	Integration
T7	16	Additive	Programming	T22	16	Disjunctive	Testing
T8	4	Conjunctive	Programming	T23	16	Additive	Programming
T9	4	Conjunctive	Programming	T24	4	Conjunctive	Programming
T10	4	Conjunctive	Programming	T25	16	Conjunctive	Programming
T11	4	Conjunctive	Programming	T26	12	Conjunctive	Programming
T12	4	Conjunctive	Programming	T27	6	Conjunctive	DB Design
T13	6	Conjunctive	Programming	T28	32	Conjunctive	Integration
T14	6	Conjunctive	Programming	T29	8	Disjunctive	Deployment
T15	8	Conjunctive	Programming	T30	12	Additive	Programming
				T31	24	Additive	Training

Table 10
Characteristics of resources available to undertake the real-world software project.

	Developer A	Developer B	Developer C	Developer D
Wage rate	€10.23	€6.25	€7.39	€5.68
Productivity rate				
Req. analysis	2.00	0.50	0.50	0.00
DB design	2.00	1.00	2.00	0.00
GUI design	2.00	0.00	2.00	0.00
Programming	2.00	1.00	1.00	0.00
Integration	2.00	0.50	0.50	0.00
Testing	0.00	0.00	0.00	2.00
Deployment	0.00	0.00	0.00	1.00
Training	2.00	0.50	1.00	0.00

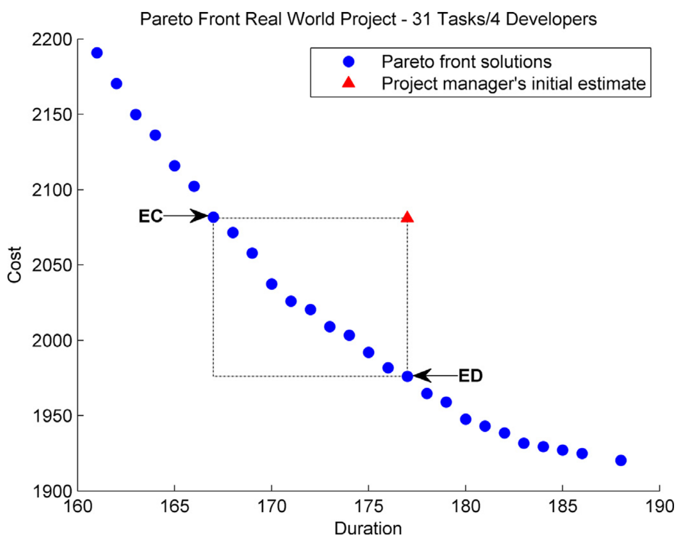

Fig. 7. Comparison of project manager's initial estimate with reference Pareto front.

allocate developers and schedule tasks so that a balance between the total duration and cost was achieved. Notably, the closest generated solutions to the project manager's estimate represent plans that also offer a more equal trade-off between the two objectives (enclosed in the dotted rectangle in Fig. 7). In other words, the project manager's estimate is not nearer either extreme, but rather closer to the midway solutions. The point on the Pareto front in Fig. 7 marked ED represents a solution whose task schedule and resource assignment yield a duration equal to the project manager's initial duration estimate. On the other hand, the point EC on the Pareto front represents a solution whose task schedule and resource assignment produce a cost equal to the project manager's initial cost estimate. The comparison of the resource assignments, costs and duration of these two points with the project manager's initial estimate for each task is presented in Table 11.

It is clear from Table 11 that by taking into account the rate of productivity of developers, the type of task interdependence and the communication overhead at task level, our optimization approach manages to assign resources and schedule tasks in a variety of ways. The generated solutions can easily be presented to the project manager through a decision support system, from which a project manager may select the most suitable allocation and

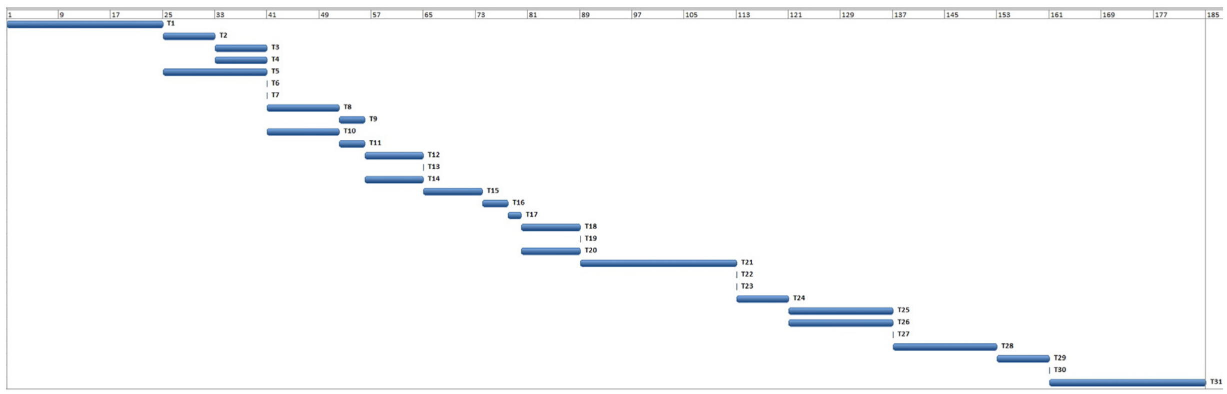

Fig. 6. Gantt chart for real-world software project constructed by the software project manager.

Table 11
Comparison of project manager's initial estimation against solutions ED and EC with respect to resource assignment, task cost and task duration.

Task	PM's initial estimation			Generated solution ED			Generated solution EC		
	Resources	Duration	Cost	Resources	Duration	Cost	Resources	Duration	Cost
T1	A	24	245.52	A	24	245.52	A	24	245.52
T2	C	8	59.12	A	8	81.84	A	8	81.84
T3	C	8	59.12	C	4	29.56	A	4	40.92
T4	A	8	81.84	A	4	40.92	C	4	29.56
T5	B	12	75.00	A	3	30.69	B	12	75.00
T6	D	4	22.72	D	4	22.72	D	4	22.72
T7	B,C	3	40.92	A,B	6	98.88	A,B	6	98.88
T8	B	4	25.00	B	4	25.00	B	4	25.00
T9	B	4	25.00	A	2	20.46	A	2	20.46
T10	C	4	29.56	B	4	25.00	C	4	29.56
T11	C	4	29.56	C	4	29.56	C	4	29.56
T12	B	4	25.00	C	4	29.56	B	4	25.00
T13	B	6	37.50	A	3	30.69	A	3	30.69
T14	C	6	44.34	A	3	30.69	A	3	30.69
T15	C	8	59.12	A	4	40.92	A	4	40.92
T16	D	4	22.72	D	4	22.72	D	4	22.72
T17	A,B,C	2	47.74	A	6	61.38	A	6	61.38
T18	B	4	25.00	C	2	14.78	C	2	14.78
T19	B	6	37.50	A	3	30.69	A	3	30.69
T20	C	6	44.34	A	3	30.69	A	3	30.69
T21	A,B	16	263.68	A	32	327.36	A	32	327.36
T22	D	8	45.44	D	8	45.44	D	8	45.44
T23	C	3	22.17	A	8	81.84	A,B	6	98.88
T24	B	4	25.00	A	2	20.46	A	2	20.46
T25	C	16	118.24	A	8	81.84	A	8	81.84
T26	B	12	75.00	A	6	61.38	A	6	61.38
T27	B	6	37.50	C	3	22.17	C	3	22.17
T28	A	16	163.68	A	16	163.68	A	16	163.68
T29	D	4	22.72	D	8	45.44	D	8	45.44
T30	B	2	12.50	A	6	61.38	A	6	61.38
T31	B,C	19	259.16	A	12	122.76	A,B,C	7	167.09
Total cost			2081.71			1976.02			2081.70
Total duration			177			177			167

schedule scheme according to his or her priorities. Furthermore, a project manager is able to generate more accurate solutions compared to ad-hoc and manual approaches with a small amount of effort. The differences between solutions are also evident when comparing each of our generated solutions with the project manager's initial estimation in regards to the overall project cost and project duration, as shown in Table 12.

As seen from the solutions that are shaded in Table 12 (i.e., the solutions enclosed in the dotted rectangle in the Pareto front of Fig. 7), compared to the estimated resource allocation and task schedule our approach was able to find a range of alternative plans that are up to approximately 6% shorter in duration for the same estimated cost or up to roughly 5% cheaper in terms of costs for the same estimated duration. The Gantt chart of the latter case (solution ED) is given in Fig. 8, where our approach generated a solution that manages to allocate developers and schedule tasks with the same project duration as the project manager's, but with a cheaper project cost. Essentially, the difference in cost is due having different combinations of developer assignments that are more cost-effective. In some tasks, the developers assigned possessed a high productivity rate and, although these developers cost more, it was still cheaper than assigning developers with a low productivity rate for a longer duration. In other tasks, the developers assigned possessed a low productivity rate and, despite the tasks taking longer, it was still cheaper than assigning developers with a high productivity rate for a shorter duration. This emphasizes the competitive nature of duration and cost, and underlines the fact that machine-based optimization activities may handle the complexity posed by this competition more efficiently compared to software project managers, irrespective of their experience and expertise. Furthermore, when we presented several of these solutions to the supervising project manager, he confirmed that in several tasks certain combinations of developers that existed both in the original allocation, as well as in the generated solutions completed the work earlier than planned, due to some of the assigned developers possessing high productivity rates in the required professions. In addition, he also confirmed that the final cost and duration of the project was much more near to the solutions generated by our approach rather than his initial estimated values. Although the percentage of reduction was relatively small due to the small size of the project, in larger and more complex projects improvements to the cost and duration estimates could be greater, proving more beneficial to software companies.

Several observations were also made regarding the type of tasks interdependence. First, the solutions generated by our approach avoided the assignment of more than one developer to certain tasks. Further examination revealed that these tasks were programming tasks that had a conjunctive interdependence type. Because the duration of a conjunctive task is determined by the lowest productivity rate in the team, our approach rejected the assignment of a team of developers to such a task in favour of assigning only one developer that possessed a high rate of productivity, thus decreasing both the duration and cost of the task. Furthermore, due to the dependency relationships, a number of programming tasks were able to be scheduled to start at the same time (for example, tasks T8, T12 and T14 in Fig. 8). In order to simultaneously avoid assignment conflicts and minimize the project's make-span, the most cost-effective solution was to assign a developer with a lower productivity rate and having the task start as soon as possible (rather than to assign one of the developers with the highest rate of productivity and forcing the start of a task to be delayed until one of those developers was available). Conversely, it was

Table 12
Difference between project manager's cost and duration estimation and each generated solution.

	Project Duration	Difference from estimated duration	Project Cost	Difference from estimated cost
PM Estimation Solution	177		2081.71	
Solution 1	161	−16	2190.84	109.84
Solution 2	162	−15	2170.37	89.37
Solution 3	163	−14	2149.90	68.90
Solution 4	164	−13	2136.27	55.27
Solution 5	165	−12	2115.80	34.80
Solution 6	166	−11	2102.17	21.17
Solution 7 (EC)	167	−10	2081.70	0.01
Solution 8	168	−9	2071.49	−9.51
Solution 9	169	−8	2057.86	−23.14
Solution 10	170	−7	2037.39	−43.61
Solution 11	171	−6	2026.03	−54.97
Solution 12	172	−5	2020.35	−60.65
Solution 13	173	−4	2008.99	−72.01
Solution 14	174	−3	2003.31	−77.69
Solution 15	175	−2	1991.95	−89.05
Solution 16	176	−1	1981.70	−99.30
Solution 17 (ED)	177	0	1976.02	−104.98
Solution 18	178	1	1964.66	−116.34
Solution 19	179	2	1958.98	−122.02
Solution 20	180	3	1947.62	−133.38
Solution 21	181	4	1943.07	−137.93
Solution 22	182	5	1938.52	−142.48
Solution 23	183	6	1931.69	−149.31
Solution 24	184	7	1929.42	−151.58
Solution 25	185	8	1927.15	−153.85
Solution 26	186	9	1924.88	−156.12
Solution 27	188	11	1920.34	−160.66

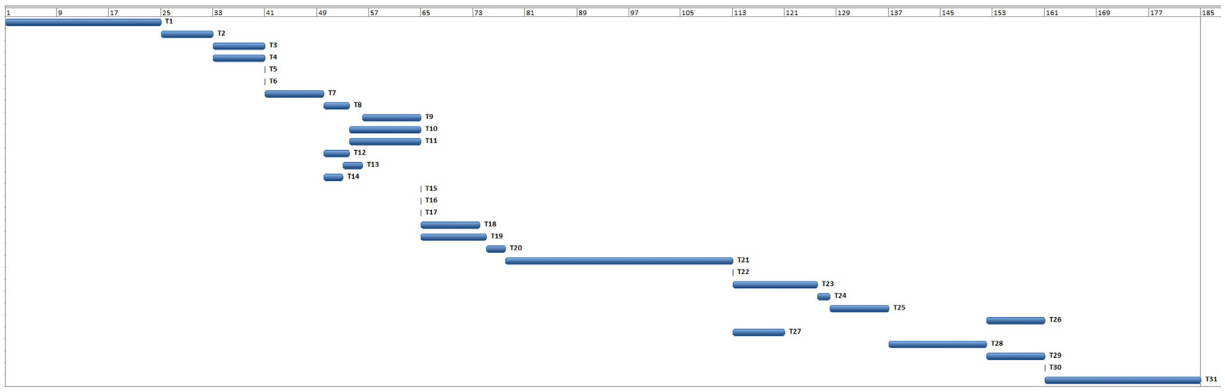

Fig. 8. Gantt chart for real-world software project corresponding to solution ED with equal duration.

observed that the solutions generated by our approach showed no preference to the number of developers assigned to additive tasks. In some cases, only one developer was assigned (the cheaper alternative), whereas in other cases, several developers were assigned (the faster alternative). These observations suggest that taking into account the type of task interdependence can prove valuable to a software project manager for his or her allocation decisions.

Overall, there are positive indications that by taking into account productivity-related attributes, our approach can provide more accurate duration and cost estimates for project planning. Furthermore, it could also provide a basis for a promising decision-support tool with which software project managers are able to efficiently select a suitable allocation of resources and task schedule that satisfies his or her criteria the most from a range of alternatives through the use of multi-objective optimization. Multi-objective optimization allows for different trade-offs between duration and cost to be examined by software project managers, which would otherwise not be possible due to the many permutations that require effort and time to produce manually. It is important, however, to examine our approach using more real-world projects from the local software industry. To this end, we have approached several development companies to provide us with project and resource data for further experimentation.

8. Threats to validity

8.1. Construct validity

Threats to construct validity that may exist concern the assumptions and simplifications made regarding the software development process. In our case, the proposed approach does not take into account each developer's degree of dedication to the tasks that he or she is assigned to. Rather, the allocation of a developer is constrained to only one task for the whole duration of that task. However, on the one hand, taking this into account was not in the scope of the current work and, on the other hand, the parallel assignment of a developer to multiple tasks requires minimal changes in the task scheduling procedure and subsequent calculation of the fitness of each individual solution. Also, we assume a maximum rate of productivity (in our case, a rate of 2.0), which in practice may be either surpassed, or contrarily, never reached. Nevertheless, any potential effect of this threat is limited, since the productivity rate values used in our approach scale relatively rather than absolutely.

8.2. Internal validity

Genetic algorithms are stochastic in nature and use various degrees of randomness in order to evolve populations and generate solutions. In order to limit this internal validity threat in our experiments, we executed each project instance 100 times for each algorithm investigated, and in addition we used the Pareto fronts generated from the solutions to apply the HV quality indicator. Furthermore, we employed statistical tests to examine if significant differences existed between the results generated by each algorithm. Also, given the random nature of genetic algorithms, the results of the experiments may be influenced by the settings and parameters chosen, such as the probabilities of mutation and crossover. In order to mitigate this, preliminary runs were carried out with various settings, which led to the use of the ones provided in Table 3.

8.3. External validity

The main threat to external validity causing a limitation in the ability to make generalizations from our findings is the fact that experimentation of our approach was carried on using randomly generated problems. To mitigate this threat, the project instances were created after discussions with several local software project managers, who were consulted in order to extract various project and developer characteristics, including number of tasks, complexity in terms of task precedence relationships, number of available developers and salary ranges. These parameters were then used to randomly generate the project instances as realistic as possible. An experiment using a real-world case study was also conducted, which showed that the solutions generated were of better quality and more realistic than compared to the schedule and allocations constructed manually by the software project manager, let alone the fact that these solutions were generated much quicker. However, further experiments using real-world software projects are necessary in order to support the results already obtained, and are planned as part of our future research. In addition, the ability of the genetic algorithms to achieve a satisfactory level of quality of solutions in a reasonable amount of time is dependent on the number of iterations that they are left to run for. This means that for larger software projects, there may be an issue of scaling, where the approach will require longer computational time in order to find better solutions. This threat however was addressed by preliminarily running the algorithms for 50,000 and 100,000 objective function executions, before finally running the algorithms for 500,000 executions. From our findings, we concluded that a significant improvement to the results is not expected by increasing the number of this value even further than 500,000 for our approach. Computation overhead may be addressed even more efficiently in cases of large projects using modules in high performance computing environments, thus executing in much less time compared to the original experiments.

9. Conclusions

This paper addressed the problem of human resource allocation and task scheduling for software development, which is one of the most challenging planning activities faced by project managers as they attempt to minimize the cost and duration of the project. We adapted the traditional RCPSP to include productivity-related attributes, focusing on the fact that developers possessing different rates of productivity carry out tasks at different speeds. We have also established that the nature of the work required to be carried out can influence how the individual contributions of developers are aggregated in terms of productivity. In addition, we also factored in the increase in the duration of a task due to communication overhead incurred while developers work together. Up until now, very few works have explored how these factors can be taken into consideration in software resource allocation and task scheduling. A multi-objective genetic algorithm approach is then applied to simultaneously minimize the duration and cost of a software project. A set of feasible and near-optimal solutions of developer allocations and task schedules were generated by evolving a population of candidate individuals with selection, recombination and mutation genetic operators. The fitness of each individual in the population was evaluated using two objective functions that assess how much time and money it will take the developer(s) assigned to complete each task based on their rate of productivity, the type of task interdependence and communication overhead. Additionally, in order to evaluate the validity of each individual, we applied two constraint functions that assess the degree to which each task has a developer assigned to carry it out and the degree to which the precedence relationships between tasks are satisfied. The benefit of using multi-objective optimization in our approach is that it is able to offer software project managers alternative near-optimal solutions rather than just one. If the criterion of a project manager is to allocate resources and schedule tasks so that the project finishes as soon as possible no matter the cost, then the choice would be made using solutions yielding a shorter make-span. Alternatively, if a project manager's goal is to allocate resources and schedule tasks so that the project costs as little as possible no matter its duration, then the selection would be based on solutions yielding lower cost. Otherwise, a project manager would look to choose a solution that balances the two criteria.

Several experiments were carried out to evaluate the performance and applicability of our approach. Sixteen synthetic software project instances of varying size and available developers were constructed and used to compare four algorithms of multi-objective genetic algorithms, namely MOCell, NSGA-II, PAES, and SPEA2. The comparison was performed using the hypervolume quality indicator, which was calculated using median values that were generated over 100 runs of each algorithm for each project instance. From the results obtained, it was clear that MOCell, NSGA-II and SPEA2 were the most dominant of the four algorithms, managing to outperform PAES in the majority of project instances. This suggests that for our approach, these algorithms are more capable of providing developer allocations and task schedules that are closer to the optimal, as well as more diverse. Additionally, thirty-six project instances were used to compare the scalability of the algorithms. Each algorithm was again executed 100 times with each project instance. By averaging the hypervolume values across both the various task sizes and available developer sizes, the results indicated that PAES was able to scale better than the rest, despite producing solutions with lower quality.

We also used data from a real-world project, the results of which provided encouraging signs on the applicability of our approach. The optimization managed to find solutions that were cheaper and with shorter make-spans compared to the initial resource allocation and task scheduling constructed by the supervising software project manager, and that these solutions were in fact more in line with the actual project cost and duration than the project manager's initial estimate. Finally, we also observed that student teams possessing varying levels of skills and competencies worked at different speeds depending on the type of task being carried out, hence enhancing our belief that task interdependence affects the way developers combine their efforts and rates of productivity, which ultimately translates into increases or decreases in task duration and cost.

There are several potential topics that can be addressed in order to improve and enhance our current approach. First, a better representation of the productivity rates of developers may help provide more accurate estimations of the time and cost of a task. Since the topic of productivity is multifaceted and complicated, a thorough investigation is required to ensure that its usage generates solutions that are as realistic as possible. Also, there is a need to explore the rates of completion of software development tasks at different levels of productivity and to examine whether a saturation point exists where a task cannot be completed in a shorter duration regardless of how highly productive a developer is. Second, apart from communication overhead there are additional factors, such as organizational, human and process factors [25], which can be incorporated within our approach to adjust the increase or decrease of the duration of a task. A recent trend in software team staffing involves the integration of the personality types of developers [54,55]. For example, personality types have been used to allocate developers to tasks and roles that are best suited to their individual traits as it has been found to affect team productivity [56]. Also, personality types have been examined in relation to issues such as job satisfaction, performance, team cohesion and social conflict [57]. All these aspects could necessitate the adjustment of the duration and subsequent cost of a task to more realistic values. A comprehensive investigation, therefore, is required in order to accurately measure and integrate these factors during resource allocation, with a suitable mechanism for project managers to prioritize their criteria. Third, our approach currently assumes that developers are available to work solely and fully on one project. Realistically, however, this is not the case as developers may be assigned to work on other projects concurrently. To improve our approach we plan to incorporate degrees of resource dedication and availability, as well as resource levelling constraints. Another intended improvement involves handling the release of developers working on divisible tasks so that they are made available again once their contribution to such a task is over rather than have them be committed for its full duration. Last, other genetic algorithm variations, as well as alternative optimization techniques can be considered, such as swarm intelligence, in order to determine whether such methods are able to provide more diverse and even nearer-to-optimal resource allocation and task schedules.

References

[1] Chang CK, Christensen MJ, Zhang T. Genetic algorithms for project management. Ann Softw Eng 2001;11(1):107–39. doi:10.1023/A:1012543203763.

[2] Pan N-H, Hsaio P-W, Chen K-Y. A study of project scheduling optimization using tabu search algorithm. Eng Appl Artif Intell 2008;21(7):1101–12. doi:10.1016/j.engappai.2007.11.006.

[3] Li C, van den Akker JM, Brinkkemper S, Diepen G. Integrated requirement selection and scheduling for the release planning of a software product. In: Sawyer P, Paech B, Heymans P, editors. Requirements engineering. REFSQ 2007: Proceedings of the 13th international working conference on requirements engineering: foundation for software quality; 2007 June 11–12; Trondheim. Norway. Berlin: Springer; 2007. 2007 p. 93–108. doi:10.1007/978-3-540-73031-6.

[4] Otero LD, Centeno G, Ruiz-Torres AJ, Otero CE. A systematic approach for resource allocation in software projects. Comput Ind Eng. 2009;56(4):1333–9. doi:10.1016/j.cie.2008.08.002.

[5] Otero CE, Otero LD, Weissberger I, Qureshi A. A multi-criteria decision making approach for resource allocation in software engineering. In: Al-Dabass D, Orsoni A, Cant R, Abraham A, editors. Computer modelling and simulation. UKSim 2010: Proceedings of the 12th international conference on computer modelling and simulation; 2010 Mar24-26. Cambridge, UK. Los Alamitos, CA: IEEE; 2010. p. 137–41. doi:10.1109/UKSIM.2010.32.

[6] Barreto A, Barros MO, Werner CML. Staffing a software project: a constraint satisfaction approach. ACM SIGSOFT Softw Eng Notes 2005;30(4):1–5. doi:10.1145/1082983.1083093.

[7] Barreto A, Barros MO, Werner CML. Staffing a software project: a constraint satisfaction and optimization-based approach. Comput Oper Res. 2008;35(10):3073–89. doi:10.1016/j.cor.2007.01.010.

[8] Antoniol G, Di Penta M, Harman M. Search-based techniques for optimizing software project resource allocation. In: Deb K, editor. Genetic and evolutionary computation. GECCO 2004: Proceedings of the 2004 genetic and evolutionary computation conference; 2004 Jun 26–30. Seattle, WA, USA. Berlin: Springer; 2004. p. 1425–6. doi:10.1007/978-3-540-24855-2_162.

[9] Jalote P, Jain G. Assigning tasks in a 24-hour software development model. Software engineering APSEC 2004: Proceedings of the 11th Asia-Pacific software engineering conference; 2004 Nov 30-Dec 3. Busan, Korea Washington, DC: IEEE; 2004. p. 309–15. doi:101109/APSEC200433.

[10] Padberg F. A study on optimal scheduling for software projects. J Softw-Evol Proc. 2006;11(1):77–91. doi:10.1002/spip.254.

[11] Chang CK, Jiang H, Di Y, Zhu D, Ge Y. Time-line based model for software project scheduling with genetic algorithms. Inf Softw Technol 2008;50(11):1142–54. doi:10.1016/j.infsof.2008.03.002.

[12] Ren J, Harman M, Di Penta M. Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. In: Cohen MB, Ó Cinnéide M, editors. Search-based software engineering. SBSE 2011: Proceedings of the 3rd international symposium on Search-Based Software Engineering; 2011 Sep 10–12. Szeged, Hungary. Berlin: Springer; 2011. p. 127–41. doi:10.1007/978-3-642-23716-4_14.

[13] Gerasimou S, Stylianou C, Andreou AS. An investigation of optimal project scheduling and team staffing in software development using particle swarm optimization. In: Maciaszek LA, Cuzzocrea A, Cordeiro J, editors. Enterprise information systems. ICEIS 2012: Proceedings of the 14th international conference on enterprise information systems; 2012 Jun 28-Jul 1. Wrocław, Poland. Setúbal: SciTePress; 2012. p. 168–71. doi:10.5220/0004001001680171.

[14] Chen W-N, Zhang J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE Trans Softw Eng. 2013;39(1):1–17. doi:10.1109/TSE.2012.17.

[15] Xiao J, Ao X-T, Tang Y. Solving software project scheduling problems with ant colony optimization. Comput Oper Res. 2013;40(1):33–46. doi:10.1016/j.cor.2012.05.007.

[16] Hapke M, Jaszkiewicz A, Slowinski R. Fuzzy project scheduling system for software development. Fuzzy Set Syst. 1994;67(1):101–17. doi:10.1016/0165-0114(94)90211-9.

[17] Callegari DA, Bastos BRM. A multi-criteria resource selection method for software projects using fuzzy logic. In: Filipe J, Cordeiro J, editors. Enterprise information systems. ICEIS 2009: Proceedings of the 11th international conference on enterprise information systems; 2009 May 6-10. Milan, Italy. Berlin: Springer; 2009. p. 376–88. doi:10.1007/978-3-642-01347-8_32.

[18] Antoniol G, Di Penta M, Harman M. Search-based techniques applied to optimization of project planning for a massive maintenance project. Software maintenance ICSM 2005: Proceedings of the 21st IEEE international conference on software maintenance; 2005 Sep 26–29. Budapest, Hungary Los Alamitos, CA: IEEE; 2005. p. 240–9. doi:101109/ICSM200579.

[19] Di Penta M, Harman M, Antoniol G. The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. Softw Pract Exp. 2011;41(5):495–519. doi:10.1002/spe.1001.

[20] Alba E, Chicano JF. Management of software projects with GAs. In: Metaheuristics. MIC 2005: Proceedings of the 6th Metaheuristics International Conference; 2005 Aug 22–26; 2005. p. 13–18.

[21] Alba E, Chicano JF. Software project management with GAs. Inform Sci. 2007;177(11):2380–401. doi:10.1016/j.ins.2006.12.020.

[22] Minku LL, Sudholt D, Yao X. Evolutionary algorithms for the project scheduling problem: runtime analysis and improved design. In: Soule T, editor. Genetic and evolutionary computation. GECCO 2012: Proceedings of the 2012 genetic and evolutionary computation conference; 2012 July 7–11. Philadelphia, PA, USA. New York: ACM; 2012. p. 1221–8. doi:10.1145/2330163.2330332.

[23] Minku LL, Sudholt D, Yao X. Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. IEEE Trans Softw Eng. 2014;40(1):83–102. doi:10.1109/TSE.2013.52.

[24] Kapur P, Ngo-The A, Ruhe G, Smith A. Optimized staffing for product releases and its application at Chartwell Technology. J Softw Maint Evol-R 2008;20(5):365–86. doi:10.1002/smr.379.

[25] Ngo-The A, Ruhe G. Optimized resource allocation for software release planning. IEEE Trans Softw Eng. 2009;35(1):109–23. doi:10.1109/TSE.2008.80.

[26] Yannibelli V, Amandi A. A knowledge-based evolutionary assistant to software development project scheduling. Expert Syst Appl. 2011;38(7):8403–13. doi:10.1016/j.eswa.2011.01.035.

[27] Yannibelli V, Amandi A. A memetic approach to project scheduling that maximizes the effectiveness of the human resources assigned to project activities. In: Corchado E, Snášel V, Abraham A, Woźniak M, Graña M, Cho S-B, editors. Hybrid artificial intelligence systems. HAIS 2012: Proceedings of the 7th international conference on hybrid artificial intelligence systems; 2012 Mar 28–30; Salamanca. Spain. Berlin: Springer; 2012. p. 159–73. doi:10.1007/978-3-642-28942-2_15.

[28] Yannibelli V, Amandi A. A diversity-adaptive hybrid evolutionary algorithm to solve a project scheduling problem. In: Corchado E, Lozano JA, Quintián H, Yin H, editors. Intelligent data engineering and automated learning. IDEAL 2014: Proceedings of the 15th international conference on intelligent data engineering and automated learning; 2014 Sep 10–12; Salamanca. Spain. Berlin: Springer; 2014. p. 412–23. doi:10.1007/978-3-319-10840-7_50.

[29] Yannibelli V, Amandi A. Project scheduling: a multi-objective evolutionary algorithm that optimizes the effectiveness of human resources and the project makespan. Eng Optim 2013;45(1):45–65. doi:10.1080/0305215X.2012.658782.

[30] Yannibelli V, Amandi A. Hybridizing a multi-objective simulated annealing algorithm with a multi-objective evolutionary algorithm to solve a multi-objective project scheduling problem. Expert Syst Appl. 2013;40:2421–34. doi:10.1016/j.eswa.2012.10.058.

[31] Steiner ID. Group processes and productivity. New York, NY: Academic Press; 1972.

[32] Di Penta M, Harman M, Antoniol G, Qureshi F. The effect of communication overhead on software maintenance project staffing: a search-based approach. Software maintenance ICSMW 2007: Proceedings of the 23rd international conference on software maintenance; 2007 Oct 2–5. Paris, France Los Alamitos, CA: IEEE; 2007. p. 315–24. doi:101109/ICSM20074362644.

[33] PMI Lexicon of Project Management Terms [Internet]. Philadelphia, PA: Project Management Institute. c2012 [cited 2015 Mar 26]. Available from: http://www.pmi.org/PMBOK-Guide-and-Standards/PMI-lexicon.aspx

[34] Curtis B., Hefley W.E., Miller S. The people capability maturity model (P-CMM) version 2.0, 2nd ed. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University; 2009 Jul. Report No.: CMU/SEI-2009-TR-003.

[35] Mäntyläa MV, Itkonena J. More testers – the effect of crowd size and time restriction in software testing. Inf Softw Technol 2013;55(6):986–1003. doi:10.1016/j.infsof.2012.12.004.

[36] Brooks FP Jr. The mythical man-month: essays on software engineering. Reading, UK: Addison-Wesley Publishing Company; 1975.

[37] Abdel-Hamid TK, Madnick SE. Software project dynamics: an integrated approach. Englewood Cliffs, NJ: Prentice Hall; 1991.

[38] Douglas MJ. The impacts of the handoffs on software development: a cost estimation model [dissertation]. Tampa: FL: University of South Florida; 2006.

[39] Holland JH. Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press; 1975.

[40] Miettinen M. Nonlinear multiobjective optimization. Norwell, MA: Kluwer Academic Publishers; 1999.

[41] Deb K. Multi-objective optimization using evolutionary algorithms. Chichester, UK: Wiley; 2001.

[42] Van Veldhuizen D.A., Lamont G.B. Multiobjective evolutionary algorithm research: a history and analysis. Wright-Patterson Air Force Base, OH: Department of Electrical and Computer Engineering, Air Force Institute of Technology; 1998 Mar. Report No.: TR-98-03.

[43] Goldberg DE, Lingle R Jr. Alleles, loci and the travelling salesman problem. In: Grefenstette JJ, editor. Proceedings of the 1st international conference on genetic algorithms; 1985 July 24–26. Pittsburgh, PA, USA. Hillsdale, NJ: Lawrence Erlbaum Associates; 1985. p. 154–9.

[44] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans on Evol Comput 2002;6(2):182–97. doi:10.1109/4235.996017.

[45] Zitzler E., Laumanns M., Thiele L. SPEA2: improving the strength Pareto evolutionary algorithm. Zurich: Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich; 2001 May. Report No.: TIK-Report 103.

[46] Knowles JD, Corne DW. Approximating the nondominated front using the Pareto archived evolution strategy. Evol Comput 2000;8(2):149–72. doi:10.1162/106365600568167.

[47] Nebro AJ, Durillo JJ, Luna F, Dorronsoro B, Alba E. Design issues in a multiobjective cellular genetic algorithm. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T, editors. Evolutionary multi-criterion optimization. EMO 2007: Proceedings of the 4th international conference on evolutionary multi-criterion optimization; 2007 Mar 5–8; Matsushima. Japan. Berlin: Springer; 2007. p. 126–40. doi:10.1007/978-3-540-70928-2_13.

[48] 2010 Standard Occupational Classification System [Internet]. Washington, DC: Bureau of Labor Statistics, United States Department of Labor. c2010 [cited 2015 Mar 26]. Available from: http://www.bls.gov/soc/classification.htm.

[49] O∗Net OnLine [Internet]. Raleigh, NC: National Center for O∗NET Development. c1998 [cited 2015 Mar 26]. Available from: http://www.onetonline.org

[50] Luna F, González-Álvarez DL, Chicano F, Vega-Rodríguez MA. On the scalability of multi-objective metaheuristics for the software scheduling problem. Intelligent systems design and applications ISDA 2011: Proceedings of the 11th international conference on intelligent systems design and applications; 2011 Nov 22–24. Córdoba, Spain Piscataway (NJ): IEEE; 2011. p. 1110–15. doi:101109/ISDA20116121807.

[51] Luna F, González-Álvarez DL, Chicano F, Vega-Rodríguez MA. The software project scheduling problem: a scalability analysis of multi-objective metaheuristics. Appl Soft Comput 2014;15:136–48. doi:10.1016/j.asoc.2013.10.015.

[52] Durillo JJ, Nebro AJ. jMetal: a Java framework for multi-objective optimization. Adv Eng Softw. 2011;42(10):760–71. doi:10.1016/j.advengsoft.2011.05.014.

[53] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evol Comput 1999;3(4):257–71. doi:10.1109/4235.797969.

[54] Stylianou C, Andreou AS. Human resource allocation and scheduling for software project management. In: Ruhe G, Wohlin C, editors. Software project management in a changing world. Berlin: Springer; 2014. p. 73–106. doi:10.1007/978-3-642-55035-5_4.

[55] Cruz S, da Silva FQB, Capretz LF. Forty years of research on personality in software engineering: a mapping study. Comput Hum Behav 2015;46:94–113. doi:10.1016/j.chb.2014.12.008.

[56] André M, Baldoquín MG, Acuña ST. Formal model for assigning human resources to teams in software projects. Inf Softw Technol 2011;53(3):259–75. doi:10.1016/j.insof.2010.11.011.

[57] Acuña ST, Gómez M, Juristo N. How do personality, team processes and task characteristics relate to job satisfaction and software quality? Inf Softw Technol 2009;51(3):627–39. doi:10.1016/j.infsof.2008.08.006.