SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2011; **41**:495–519
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1001

The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study

Massimiliano Di Penta[1,*,†], Mark Harman[2] and Giuliano Antoniol[3]

[1]*Department of Engineering, University of Sannio—Palazzo ex Poste, Via Traiano, 82100 Benevento, Italy*
[2]*Department of Computer Science, King's College London—Strand, London WC2R 2LS, U.K.*
[3]*Department of Génie Informatique, École Polytechnique de Montréal, Montréal, QC, Canada*

SUMMARY

Allocating resources to a software project and assigning tasks to teams constitute crucial activities that affect project cost and completion time. Finding a solution for such a problem is NP-hard; this requires managers to be supported by proper tools for performing such an allocation. This paper shows how search-based optimization techniques can be combined with a queuing simulation model to address these problems. The obtained staff and task allocations aim to minimize the completion time and reduce schedule fragmentation. The proposed approach allows project managers to run multiple simulations, compare results and consider trade-offs between increasing the staffing level and anticipating the project completion date and between reducing the fragmentation and accepting project delays. The paper presents results from the application of the proposed search-based project planning approach to data obtained from two large-scale commercial software maintenance projects. Copyright © 2011 John Wiley & Sons, Ltd.

Received 10 August 2009; Revised 29 January 2010; Accepted 29 June 2010

KEY WORDS: search-based software engineering; genetic algorithms in software engineering; software project management

1. INTRODUCTION

Project planning constitutes a vital task for many software engineering activities, from development to maintenance. Poor planning can cause unacceptable costs and delays, giving the project time and financial constraints. It is well known that poor planning, together with wrong cost estimation, is one of the main causes of many software project failures. Of course, there are existing tools such as the Project Evaluation and Review Technique (PERT), the Critical Path Method (CPM), Gantt diagrams, and Earned Value Analysis, all of which can help to plan and track project milestones. While these tools and techniques are important, they are less helpful when solving important problems such as (i) determining the staffing needed to complete the project within a given date; (ii) creating working teams with the available developers; and (iii) allocating atomic tasks to the different teams.

To achieve these objectives, we first need to organize the work (i.e. the project) in atomic, work packages (WPs), often derived from the project Work Breakdown Structure (WBS). Then, WP development/maintenance effort is estimated, using one of the many techniques available (such as COCOMO [1] or analogy estimation [2]). Subsequently, staffing can be determined using a queuing-simulation approach, as proposed by Antoniol *et al.* [3]. The project schedule can be

*Correspondence to: Massimiliano Di Penta, Department of Engineering, University of Sannio—Palazzo ex Poste, Via Traiano, 82100 Benevento, Italy.
†E-mail: dipenta@unisannio.it

further optimized by (i) determining the distribution of developers into teams and (ii) properly assigning WPs to teams. The two factors are clearly interwoven. Given a fixed distribution of developers into teams, there exists a WP assignment to teams that minimizes the time required to carry out the project. This can also be thought of as the optimal order in which the WPs flow into a queuing network that models the development or maintenance process [3]. Such resource allocation problems are instances of the 'bin packing problem', the solution of which is known to be NP-hard [4] and for which, search-based techniques are known to be effective [5]. For a large software project, the space of possible project planning solutions is far too large to search manually or to adopt exhaustive enumeration of solutions to locate the optimal project organization. The problem of simply determining the optimal ordering of n WPs has search space size $n!$, which is clearly prohibitively large for non-trivial projects.

This paper explores the way in which search techniques can be combined with techniques from queuing theory to produce optimal or near-optimal solutions to software project planning. Specifically, the paper addresses problems associated with the determination of the order in which to process WPs, and their allocation to teams of developers. Search-based techniques are not guaranteed to converge to a global optimum within available time; nevertheless they are able to find near-optimal solutions that are often close enough to the best possible project configuration. In our case, the manager is interested to complete the project within a given deadline or to deliver the product to the customer in a time that is short enough not to conflict with his/her other work, and to satisfy the customer delivery requirements.

The decision whether a particular solution can be considered acceptable is left to the project manager; the approach simply provides decision support by allowing for exploration of the trade-offs and inherent tensions between potential solutions. In other words, the approach aims to aid the manager in her/his scheduling and staffing task, and not to replace her/him. In many software projects, the manager's task is to balance conflicting goals/objectives such as staffing level, completion time or resource utilization, rather than finding a global optimum for one of these objectives. The approach can be applied to both development and maintenance projects, and accounts for several realistic situations that typically occur in real world software project scenarios, such as

1. *Dependences*. In particular, we deal with inter-dependences between WPs. Any WP that has a dependence cannot be handled until the WP on which it depends has been processed.
2. *Schedule fragmentation*. This is a consequence of inter-dependences. If a WP is assigned to a team and is ready to be processed, but WPs upon which it depends are yet to be processed, the team is potentially idle, awaiting completion of pre-requisites.
3. *Conflicting objectives*. Shorter project completion time can be achieved with higher staffing costs and vice versa. The approach provides the manager with a Pareto front [6] of solutions optimizing conflicting objectives.
4. *Need for specialized developers*. In many cases it happens that WPs need to be dispatched to teams composed of developers specialized in specific domains or knowledgeable about a given technology. The approach accounts for staff teams of specialized developers and addresses the problem of determining the WP allocation to these specialized teams.

In order to validate the techniques introduced in the paper, empirical results are presented from two large-scale case studies of maintenance projects from two large software development organizations, specifically a Y2K remedial work project (*Project A*) and a multi-platform evolution project (*Project B*). *Project B* required, overall, less effort than *Project A*, while its development activities were constrained by dependences among WPs. The empirical study compares the performance of different heuristics, namely Genetic Algorithms (GAs), Stochastic Hill Climbing (SHC), and Simulated Annealing (SA).

More specifically, the primary contributions of this paper are as follows:

1. The paper introduces an approach to search-based software project management that caters for the above-mentioned four issues of dependences, fragmentation, conflicts, and specialization. These four aspects are highly prevalent in software projects and hence any optimization-based approach to project management needs to pay particular attention to them if it is to be useful

DOI: 10.1002/spe

as a decision support tool for software intensive projects. The approach is also novel because it is the first to use a Pareto-optimal approach to balancing conflicting concerns in software project management.
2. The paper adapts several algorithms for search-based optimization to apply them to these problems. The algorithms used in the paper are SA, single and multi-objective GAs and hill climbing.
3. The paper presents the results of experiments on data from two large real-world software projects, which assess the performance of each of the algorithms, and the effects of different staffing levels, fragmentation and employment of specialist developers.

The optimization algorithms and methods used in this paper are complex to be understood by project managers. However, the sophistication of the proposed approach can be easily 'wrapped' to hide this complexity; managers need only input information which they are familiar with (e.g. staffing level, developer expertise, constraints between WPs and so on). The results the manager receives are also easily intelligible without any knowledge of the techniques being used. This 'wrapability' must be an important feature of any approach to search-based software project management, since the decision makers cannot be assumed to have interest or skills in search-based optimization.

The remainder of the paper is organized as follows. Section 2 discusses the related work on the use of search-based techniques to solve software engineering problems, and the related literature on project scheduling and planning. Section 3 describes the proposed search-based project planning approach, while Section 4 details the way in which search-based optimization techniques can be used to find a solution to the project scheduling and staffing problem. Section 5 presents the results of a series of experiments that evaluate the proposed techniques on data from the two commercial software projects. Finally, Section 6 concludes.

2. RELATED WORK

Search techniques have previously been applied to scheduling problems in general with good results as described by Davis [7]. The mathematical problem encountered is, as described by the author, the classical, NP-hard, bin packing, or shop-bag problem. A survey of approximated approaches for the bin packing problem is presented by Coffman *et al.* [8].

A survey of the application of GAs to solve scheduling problems has been presented by Hart *et al.* [9]. Kolisch and Hartmann [10] also surveyed different techniques for project scheduling, although their study was not focused on software engineering. Our work shares with this previous work the finding that techniques, such as GAs and SA, represent the best search-based techniques in terms of performance for this kind of problem. This may be an indication that Search-based Software Project Planning (SBSPP) may benefit from results from the more general literature on project scheduling.

However, these previous results have been obtained from non-software projects. It is only more recently that researchers have turned their attention to the special problems encountered by software project managers and the ways in which search-based optimization techniques can be applied to help to provide mechanisms for decision support.

Surveys of the general area of Search-based Software Engineering (SBSE) that contain a treatment of SBSPP can be found elsewhere [11–13], this section focuses specifically on the development of SBSPP as a subarea of activity within SBSE and the relationship between this previous work to that in the present paper.

Chang *et al.* [14–17] were the first to publish on SBSPP [14], with their introduction of the Software Project Management Net (SPMNet) approach for project scheduling and resource allocation, which was evaluated on simulated project data. Aguilar-Ruiz *et al.* [18, 19] also presented early results on evolutionary optimization for SBSPP, once again evaluated on synthetically created software project data. Like Aguilar-Ruiz *et al.* our approach seeks to inform and assist the project manager, and not to replace the crucial human element in decision making about project plans.

DOI: 10.1002/spe

The allocation of teams to WPs in SBSPP can be thought of as an application of a bin packing problem [8]. Motivated by this observation, Alba and Chicano [20, 21] applied search algorithms to software projects to seek to find allocations that achieve earliest completion time (tightest packed bins, where bins are teams and WPs are the items to be packed). The present authors also worked on a similar bin packing formulation [22] upon which the present paper builds. Alba *et al.* [23] were the first to experiment with parallel implementations of SBSE techniques for SBSPP, thereby exploiting the potential of parallel computation for improved efficiency.

In the past three years, there has been a significant upsurge in interest in SBSPP, with many different SBSE techniques being applied to the problem. For example, a scatter search has been recently applied [24] to the problem of minimizing project completion duration. Scatter Search was also used by Alba *et al.* [23] but it is a technique not otherwise much used in SBSE work. Also, several recent papers (published in 2008) [25–27] present results for formulations of software project planning, evaluated using synthetic data.

The present paper is a version of our previous work [22], extended to handle developer specialization, dependences, and fragmentation, and also extended to present more empirical results and evaluation and additional algorithms (notably the use of a Pareto-optimal approach for handling conflicts). The use of queuing simulations also builds on the present authors' previous work (see Antoniol *et al.* [3]).

None of the previous work on allocation of staff to WPs has used a Pareto-optimal approach. Also, to the authors' knowledge the evaluation previous work on these problems has been based solely on synthetic rather than real data. For the related topic of search-based cost estimation for software projects [28–30], real data is routinely used in evaluation, but for SBSPP there is comparatively little real-world data available. Synthetic data is very useful for experimenting with search-based algorithms under controlled conditions; it facilitates exploration of scalability and robustness. However, real-world data additionally allows the researcher to address questions, such as the effects of different staffing levels, fragmentation effects, and employment of specialist developers, as we do in the present paper.

Similar to the present paper, the authors' recent work on risk reduction for SBSPP [31] presents an evaluation using real-world data and also adopts a Pareto-optimal approach. However, this other recent work [31] is concerned with finding a robust project schedule and does not cater for the issues of fragmentation and developer skill assignment.

Queuing theory was also applied by Ramaswamy [32] to model software maintenance projects. Simulations of a software maintenance process were performed by Podnar and Mikac [33] with the purpose of evaluating different process strategies rather than staffing the system. Recently, Bertolino *et al.* [34] proposed the use of a performance engineering technique, based on the use of queuing models and UML performance profiles, to aid project managers for decision making related to the organization of teams and tasks. We share with them the idea of using queuing networks to model software processes and to support managers in their choices. In our case, the use of search-based techniques suggests to managers the configuration able to minimize the completion time and maximize the efficiency in resource usage, also considered an important issue by Bertolino *et al*.

Recently, Gutjahr *et al.* [35] and Stummer *et al.* [36] formulated the portfolio selection scheduling and staffing problem as a nonlinear mixed integer optimization. The proposed objective function aggregates two terms: a per project gain and a global company efficiency improvement over the scheduled time span. Employees possess skills, abilities modeled as competencies; competencies are quantified as real values possibly changing over time. Projects are made up by tasks and each task is decomposed into one or more WPs. A WP is a logical unit requiring a given, deterministic and known, effort of a single competence. Then, the problem is solved using different heuristics, namely Ant Colony Optimization (ACO), and GAs combined with a problem-specific greedy technique. Results of a study conducted on data from an e-commerce project showed that GAs are slightly superior to ACO except in some cases.

Several commonalities can be found with the present work. We share with Gutjahr *et al.* [35] and Stummer *et al.* [36] the assumption of atomic tasks, the fact that a WP is organized around

DOI: 10.1002/spe

a single precise skill and high level ideas. However, they tackle a different problem, the portfolio management problem whereas we assume that the project is already assigned. In our formulation we explicitly model two contrasting objectives: the project completion time and the fragmentation. Completion times and fragmentation are only implicitly managed by Gutjahr *et al.* and Stummer *et al.* via constraints. At a higher level, we sought to determine the Pareto front and let the project manager decide the final project configuration, whereas Gutjahr *et al.* and Stummer *et al.* compute a solution that meets the constraints.

Other than search-based optimization techniques, other approaches are also used for (software) project scheduling. For example, Constraint Logic Programming (CLP) was used by Barreto *et al.* [37]. However, the focus of their work is different from ours: they aimed at assigning maintenance requests to the most qualified team in terms of skills, or to the cheapest team, or to the team having the highest productivity. To this aim, they assumed the existence of a relationship between completion time and developer skills. This, however, was not empirically supported by data from real projects. Instead, the authors showed with a controlled experiment that managers were able to benefit for the tool to schedule a (fictitious) project better and faster. Results of such an experiment indicates that automatic project scheduling approaches can be a valuable support for managers, thus approaches addressing different objectives, like ours, are worth being investigated.

Issues connected with empirical studies of software process simulation modeling are also discussed by Raffo *et al.* [38], with particular reference to the estimate of simulation parameters from real-world data, and to compare the actual results with the model's results. Abdel-Hamid [39] published an approach based on system dynamics modeling to verify the degree of interchangeability of men and months [40]. Results from the analyzed case study do not fully support Brooks' law. Nevertheless, in case there is communication overhead, the proposed search-based approach can still be applied, as shown in a companion paper [41], where the authors discuss the effect of different communication overhead models on the project planning (and resulting completion time) obtained by means of the proposed search-based approach.

3. THE SEARCH-BASED PROJECT PLANNING APPROACH

The approach presented in this paper aims to find (near) optimal solutions for the following problem:

> *Given a set of WPs—having some dependence constraints—and an available pool of N developers, create teams of developers and assign WPs to these teams so that the project completion time is* (*near*) *minimized. Also, the approach shall* (*near*) *minimize the schedule fragmentation, i.e. the total number of days in which developers remain inactive because the WPs they have been assigned depend on other WPs still to be processed. Finally, if each WP requires a particular expertise, the approach shall staff teams composed of specialized developers, so that each WP is assigned to a team having the proper expertise.*

In the following we provide a formal statement of the above formulated problem. Then, it is described how a problem solution can be represented, and how the quality of a solution (the fitness function) can be evaluated.

3.1. Problem statement

Let us consider a maintenance or development project having assigned N developers $D \equiv \{d_1, d_2, \ldots, d_N\}$ and composed of M atomic WPs $W \equiv \{w_1, w_2, \ldots, w_M\}$. Each WP w_i is characterized by an estimated development/maintenance effort e_i and requires a particular expertise ex_i in a set of expertises $Ex \equiv \{ex_1, ex_2, \ldots, ex_P\}$. The function $ExW : W \rightarrow Ex$ given one WP w_i assigns its expertise $ExW(w_i)$.

Each software developer possesses a set of expertise competences in Ex. The function ExD: $D \rightarrow 2^{Ex}$, given developer d_j returns her/his set of expertise $ExD(d_j)$. Furthermore, we consider

DOI: 10.1002/spe

a set of dependences between WPs $Dep \equiv \{dp_1, dp_2, \ldots, dp_Q\}$ each one having the form dp_l: $w_i \rightarrow w_j$, i.e. w_i needs to be completed before w_j can be started. Notice that in this model cyclic dependences, in theory, may exist. However, we do not model the case in which two or more WPs have to be worked out simultaneously, in parallel. Thus, prior to applying the approach, it is necessary to make the dependency graph acyclic by removing cyclic dependences or collapsing loops into a single WP.

The manager's task is to determine a set of L teams $\mathscr{T} \equiv \{T_1, T_2, \ldots, T_L\}$, an assignment of WPs to teams and a suitable order of assigned WPs such that dependences are satisfied while minimizing completion time and fragmentation.

To support the manager in her/his task we assume that teams in $\mathscr{T}$ must have the following four properties:

1. $\bigcup_i T_i \subseteq D$, i.e. teams are composed solely of developers in D;
2. $\forall i, j \wedge i \neq j T_i \cap T_j = \phi$, i.e. teams are non-overlapping;
3. $\forall T_j \in \mathscr{T} : \bigcap_{d_i \in T_j} ExD(d_i) \neq \phi$, i.e. developers in one team must share a common set of expertise competences;
4. $\forall w_i \in W \exists T_j \in \mathscr{T} | ExW(w_i) \in [\bigcap_{d_l \in T_j} ExD(d_l)]$, i.e. in order to complete the project, the manager must allocate teams so that the required expertise is covered by at least one team.

To model the manager's choices, any approach supporting project scheduling should also provide a function $MwT : W \rightarrow \mathscr{T}$ such that given a WP w_i it produces its assignment $MwT(w_i)$ to a team. We require that MwT defines a partition $WPT_1, WPT_2, \ldots, WPT_L$ over W—i.e. it assigns each WP to one and only one team—such that the assignment is compliant with expertise constraints. More formally, we require that for each w_j, the constraint $ExW(w_j) \in [\bigcap_{d_l \in MwT(w_j)} ExD(d_l)]$ must be satisfied. In the following, to simplify the notation, let us denote by $MwT(W, T_j)$, the set WPT_j of WPs assigned to T_j. For each set WPT_i, the manager must define a permutation P_i so that dependences in Dep are satisfied while completion time and fragmentation are minimized.

To express and verify dependences, we observe that given $\mathscr{T}$, $WPT_1, WPT_2, \ldots, WPT_L$ and the set of permutations $P_1(WPT_1), \ldots, P_L(WPT_L)$, it would be possible to determine when each WP will start and when it will be completed. As will be described in Section 3.2, this is done by using queuing simulation. That is, assume that the WP w_i is assigned to team T_{a_i} having a workload WP_{a_i} ordered as defined by the permutation P_{a_i}, then the queuing simulation computes the start time and the completion time for w_i, referred as $tb(w_i, T_{a_i}, P_{a_i})$ and $te(w_i, T_{a_i}, P_{a_i})$, respectively.

Dependence constraints can be described in terms of tb and te. That is, a dependence $dp_l : w_i \rightarrow w_j$ is mapped into the additional problem constraint as follows. Suppose that WP w_j is assigned to T_{a_j}, then dependences are preserved provided we have that: $tb(w_i, T_{a_i}, P_{a_i}) \geq te(w_j, T_{a_j}, P_{a_j})$.

Overall we must determine L, $\mathscr{T}$, MwT (and thus $WPT_1, WPT_2, \ldots, WPT_L$) the family of permutations $P_1, \ldots, P_L$ so that queuing simulation can compute two functions representing the objectives of our optimization problem:

- $f_1 = Ct(L, T_1, \ldots, T_L, MwT, P_1, \ldots, P_L)$: *completion time* of the overall project, i.e. time from when the first WP is sent into the queue to when the work on the last WP has been completed;
- $f_2 = Frag(L, T_1, \ldots, T_L, MwT, P_1, \ldots, P_L)$: *fragmentation*, i.e. the total amount of idle person months in the schedule, due to the need for completing a WP before other WPs in the queue could be processed. The fragmentation issue is discussed in detail in Section 3.4.1.

Using this notation, it is possible to define the project allocation optimization problem formally as follows:

$$\min_{L, T_1, \ldots, T_L, MwT, P_1, \ldots, P_L} (f_1, f_2)$$

$$such\ that$$

$$\forall T_i, \quad T_j \in \mathscr{T} : T_i \cap T_j = \phi \qquad \text{Disjoint teams}$$

DOI: 10.1002/spe

$$\forall T_j \in \mathscr{T} : \bigcap_{d_i \in T_j} ExD(d_i) \neq \phi \quad \text{Shared expertise} \tag{1}$$

$$\forall w_i \in W \, \exists T_j \in \mathscr{T} \mid ExW(w_i) \in \left[\bigcap_{d_l \in T_j} ExD(d_l) \right] \quad \text{Available expertise}$$

$$\forall (dp_l : w_i \rightarrow w_j) \in Dep : tb(w_i, T_{a_i}, P_{a_i}) \leq te(w_j, T_{a_j}, P_{a_j}) \quad \text{Dependences met}$$

$$\forall T_i, T_j \in \mathscr{T} : \nexists w_k \in W \mid MwT(w_k, T_i) \cap MwT(w_k, T_j) \neq \phi \quad \text{One team per WP}$$

$$\forall w_i \in W : ExW(w_i) \in \bigcap_{d_l \in MwT(w_i)} ExD(d_l) \quad \text{Appropriate expertise}$$

3.2. *Modeling the maintenance/development process as a queuing system*

We start by assuming that all WPs belong to the same category, thus no particular expertise is required from the developer. Then, in Section 3.5, we relax this constraint, describing how teams of specialized developers will be handled.

To solve the problem defined in Section 3.1 using a search-based approach, the process of software development/maintenance is modeled as a queuing system [3]. However, we do not use the approach previously used for this [3]. Rather we build a queuing simulator that simulates how WPs—ordered as determined by a possible solution of our problem—flow into teams (or wait in the queue for an available team), and are then processed.

As shown in Figure 1, a queuing system can be described as 'customers' arriving for service (WPs in our case), waiting for service (development or maintenance task) if it is not immediate, and leaving the system after being served (by a team of developers, referred as 'servers' in queuing theory). Further details on queuing theory can be found in the book by Gross and Harris [42].

In the context of software project planning, given a possible problem solution, i.e. a distribution of developers among teams, and a permutation over the WP ordering, the queuing simulation is used to compute the project completion time. Observe that the order in which WPs enter into the queue determines the team to which the WP is assigned (i.e. the first available one).

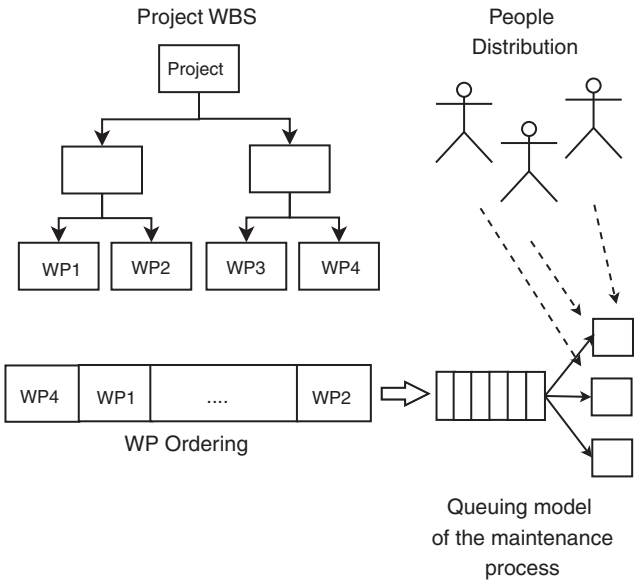

Figure 1. Project scheduling: the queuing model.

DOI: 10.1002/spe

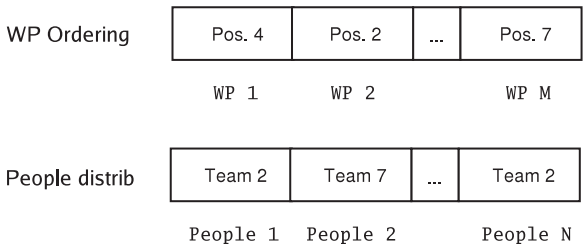

Figure 2. Solution representation.

A search-based project planning approach includes the following components:

1. The heuristic for solution generation. For instance, those described in Section 4. The heuristic approach generates a solution.
2. The fitness of the objective(s) to be optimized. For instance, solution completion time and, if necessary, other dependent variables such as schedule fragmentation (see Section 3.4.1). These objectives are evaluated by means of a queuing simulation, and used by the heuristic to guide the generation of subsequent solutions.

The search-based optimization process iterates these two steps until the heuristic stopping criteria is satisfied. That is, the process continues to generate and evaluate candidate solutions until either a set of acceptable solutions are discovered or the process is 'timed out'.

3.3. Solution representation

Consider a project modeled as a single node queuing system. Given this, a solution to the project planning problem can be represented in a 2-array data structure as shown in Figure 2. The first array represents the WP ordering in the incoming queue; it is an M-sized array (where M is the number of WPs), and the value of an entry indicates the position of the WP in the incoming queue, for a single-queue/multi-server queuing system. As shown in Figure 2, the second array is an array of size N, where N is the number of developers available for the project. Each value of the array indicates the team to which a developer is assigned. Section 3.5 explains how the representation is extended to allow for specialized teams.

3.4. Evaluating a solution: the fitness function

As explained in Section 3.2, the quality of a solution—referred as 'fitness function'—in metaheuristics is quantified as the completion time and fragmentation, estimated by the queuing simulator. Basically, the simulator picks incoming requests from the queue and sends them to available teams, and for each one of them, given s the size of the team to which the request is dispatched and e the estimated effort for the WP, the time necessary to complete the WP will be considered as $t = e/s$.

As discussed in Section 5.5, due to Brooks' law [40], this could be an optimistic assumption, i.e. the time required to complete a task might not be just the effort divided by the team size, because for example of overhead due to team member communication, or to training activities needed when adding a new team member to an ongoing activity. Truly, this does not lessen to the approach generalizability, and is reasonable given the nature of the projects we used as case studies. In fact, the model can be generalized to situations in which Brooks' law applies by the simple introduction of a nonlinearity factor. Generalized models taking care of communication overhead are discussed elsewhere [41]. The overall project completion time will be equal to the time elapsed from when the first WP is sent into the system to the time when the last WP has been completed.

The completion time depends upon the particular queuing configuration (determined by varying the distribution of developers across teams) and the given WP ordering. However, the search algorithm must be able to handle dependences between WPs. Among the different ways of handling

DOI: 10.1002/spe

constraints in metaheuristic search algorithms [43, 44], we chose to repair solutions violating constraints, i.e. searching for neighbor acceptable solutions:

1. WPs are seeded into the queue in the order specified by the first line of the representation.
2. Every time a team is available, the dispatcher tries to assign it the WP in front of the queue. In case such a WP cannot be handled yet because of a dependence, the schedule searches back in the queue until it finds a WP that can be handled.
3. If the dispatcher is able to find a WP, it will be assigned to the available team. If not, this means that no assignment is currently possible, because one of the WPs currently under work needs to be completed before any of the waiting WPs can be assigned. In that case, simulation proceeds leaving the available team 'idle'.

It is important to remark that, given any possible schedule, this always corresponds to a WP ordering in the queue. In fact, the combination of WP ordering, WP required effort, team staffing level, WP dependencies, and the above-mentioned rules to dispatch WPs to teams determines the assignment of WPs to teams.

3.4.1. Dealing with schedule fragmentation—multi-objective optimization. As described before, when there are dependences between WPs, there might be cases for which the schedule generated to minimize the project completion time requires that some teams remain inactive for a period of time. While it would be acceptable that a team starts its work later or finishes earlier (in that case staff can be assigned to other projects), it is an additional goal to avoid—or at least to limit—schedule fragmentation. That is, we seek to limit the schedule fragmentation, defined as the total number of idle days for teams that have been already assigned to the project. Although idle days may not always be a problem—e.g. they could be used for training purposes—they, nonetheless, represent an additional, unwanted cost for the project and a potential waste of resources, especially if this happens for larger teams. On the other hand, given a distribution of developers across teams, there may not exist a schedule with no fragmentation. Furthermore, even if there did exist a fragmentation-free schedule, it may be unacceptably expensive.

Reducing completion time and schedule fragmentation are two potentially conflicting objectives, since solutions achieving a shorter completion time can exhibit higher schedule fragmentation, and vice versa. Let us consider the example in Figure 3, where (i) for each WP the effort needed is of one person day, (ii) both Teams A and B are composed of one developer only, and (iii) WPs have dependences as illustrated in Figure 3(a) (WP1 $\rightarrow$ WP3 indicates that WP1 must be completed before WP3 starts). In this example, we can have a schedule like the one in Figure 3(b), where the completion time is of three days, however Team B suffers from a schedule fragmentation, or the one in Figure 3(b), where there is no fragmentation but the completion time is four days whereas Team B works just for one day.

For this kind of a situation, instead of having one problem solution achieving a compromise between the two objectives, the manager might want to choose among different possible solutions. To this aim, we can use Pareto optimality. With Pareto optimality, one cannot measure to what *extent* a solution is better than another. However, it is possible to determine *whether* a solution for a multi-objective problem is better than another. That is, given two solutions x_1 and x_2:

$$F(x_1) < F(x_2) \Leftrightarrow \exists j \,|\, f_j(x_1) < f_j(x_2) \wedge \forall i f_i(x_1) \leq f_i(x_2) \tag{2}$$

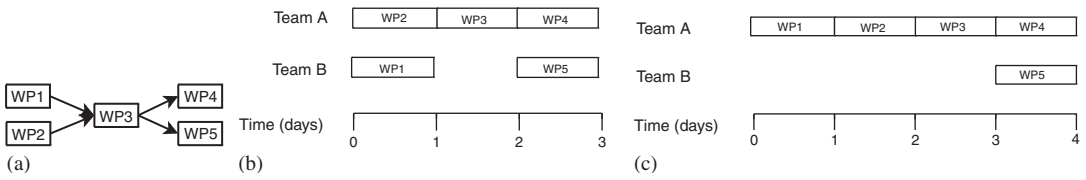

Figure 3. Example of fragmented schedule and of non-fragmented schedule with a higher completion time: (a) WP dependences; (b) schedule with fragmentation; and (c) schedule without fragmentation.

DOI: 10.1002/spe

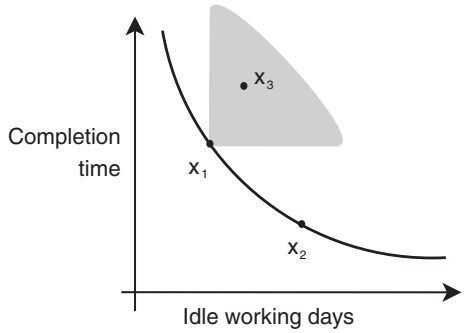

Figure 4. Multi-objective project planning: Pareto front and dominating/dominated solutions.

where $i \in O$, the set of all problem objectives. In other words, one solution is better than another if it is better for at least one objective (fitness function) and no worse for all the others. If this does not happen—e.g. a solution is better for one objective, but worse for another—then two solutions are incomparable, and the user might have to decide whether to give priority to one objective or the other.

When searching for solutions using Pareto optimality, the search produces a set of *non-dominated* solutions (referred as a 'Pareto front') rather than as a single solution. Non-dominated means that there do not exist other solutions that are better in terms of all the objectives. For example, in Figure 4 solutions x_1 and x_2 belong to the Pareto front: whereas x_1 is better in terms of schedule fragmentation, x_2 is better in terms of completion time. Instead, x_3 belongs to the dominated (shaded) area of x_1, being worse than x_1 in terms of both objectives. Details on multi-objective optimization can be found in [6, 44].

As, different from the completion time, the fragmentation is not a regular function, and since when solving the problem formulated in Section 3.1 a heuristic would attempt to start dispatching a WP to a team as soon as possible—this, in the scheduling theory, is referred to as a 'semi-active schedule' [45]—there is no guarantee that the solution found is the global optimum.

Another scenario in which multi-objective optimization can be useful in this context is when one wants to balance between minimizing the completion time and having a lower staffing level. In this case we will have a Pareto front like the one in Figure 4, where, for decreasing staffing levels (x-axis), we can have increasing project completion time (y-axis).

3.5. Employing specialized developers

Until now we have assumed that developers/maintainers are sufficiently skilled to be allocated to any task. When this is not the case, the single-queue model shown in Figure 1 becomes a multi-queue system, where each queue dispatches requests related to a team composed of developers having expertise for a particular specialization (see Figure 5(a)). Although a developer can have more than one expertise, in this paper we assume—in the context of a project—to assign a developer to a single specialization only. The future work will deal with more complex scenarios. For example, by creating teams corresponding to the various possible combinations of expertise and dynamically changing team composition on demand.

The problem representation (Figure 5(b)) is similar to the one for a staged development/maintenance process. WP ordering is once again represented as an array, where WPs are labeled according to the specialization. There is no need to have separate ordering since a permutation of such an array also determines the order of WPs flow in the separate, specialized queues. The allocation of staff into a specialized team is represented by means of P arrays, one for each expertise. Also in this case, as explained in Section 3.4 the schedule is properly repaired to handle dependences, which can occur between WPs related to the same or different specialization.

DOI: 10.1002/spe

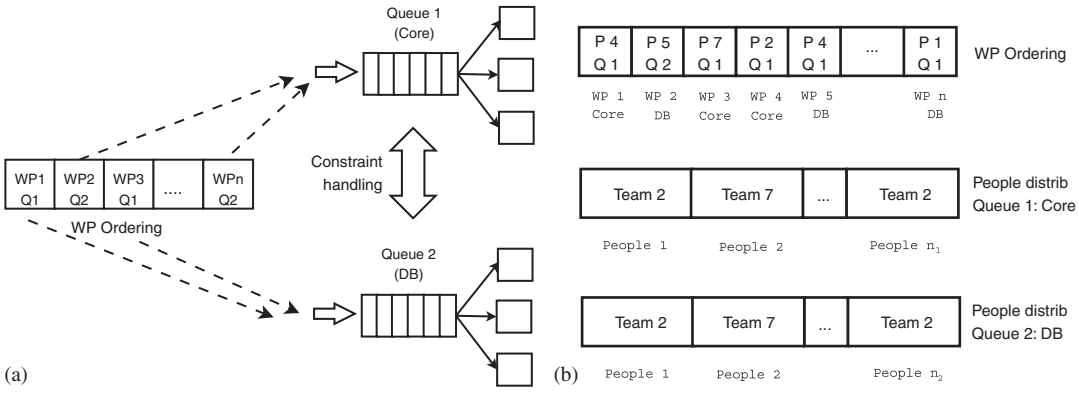

Figure 5. Schedule with specialized queues: (a) multi-queue approach and (b) representation.

4. SOLVING THE PROJECT STAFFING PROBLEM

Once the project planning problem has been modeled, a representation provided and a fitness function described, we can use different search-based algorithms to solve it. We applied GAs, SHC, and SA in order to experiment with those optimization techniques most widely used in SBSE (these techniques account for those used in over 80% of all SBSE publications according to a recent comprehensive survey [13]). SHC and SA will be used as single-objective optimization techniques, with the aim of (near) minimizing the project completion time only. As detailed in Section 4.1.2, GAs will be used both as single objective technique—and compared with SHC and SA—and multi-objective techniques with the aim of considering both completion time and fragmentation as (potentially conflicting) objectives.

4.1. An overview on the search-based algorithms used

Below we briefly describe the three algorithms we adopted to solve the scheduling and staffing problem. Further details can be found in books by Goldberg [46] or by Michalewicz and Fogel [44].

A GA may be defined as an iterative procedure that searches for the best solution of a given problem among a population having a constant or variable size, and represented by a finite string of symbols, the *genome*. The search starts from an initial population of individuals, often randomly generated. At each evolutionary step, individuals are evaluated using a *fitness function* and selected using a *selection mechanism*. High-fitness individuals will have the highest reproduction probability. The evolution (i.e. the generation of a new population) is affected by two genetic operators: the *crossover operator* and the *mutation operator*. The crossover operator takes two individuals (the *parents*) of the old generation and exchanges parts of their genomes, producing one or more new individuals (the *offspring*). The mutation operator has been introduced to prevent convergence to local optima; it randomly modifies an individual's genome (e.g. by flipping some of its bits when the representation is a binary string).

SHC is a local search method, where the search proceeds from a randomly chosen point (solution) in the search space by considering the neighbors (new solutions obtained by mutating the previous solution) of the point. Once a fitter neighbor is found this becomes the current point in the search space and the process is repeated. If, after mutating a given x number of times no fitter neighbor is found, then the search terminates and a maximum has been found (by definition). To avoid local maxima, the hill climbing algorithm is restarted multiple times from a random point.

SA [47], like hill climbing, is a local search method. However, SA has a 'cooling mechanism' that initially allows moves to less fit solutions if $p < m$, where p is a random number in the range $[0 \ldots 1]$ and m a value that decays ('cools') at each iteration of the algorithm according to the following law $m = e^{\Delta\text{fitness}/T}$, where T (temperature) is $T = T_{\max} \cdot \mathrm{e}^{-j \cdot r}$ ($T_{\max}$ is the starting temperature, r is the *cooling factor*, j the number of iterations), and Δ fitness is the difference

DOI: 10.1002/spe

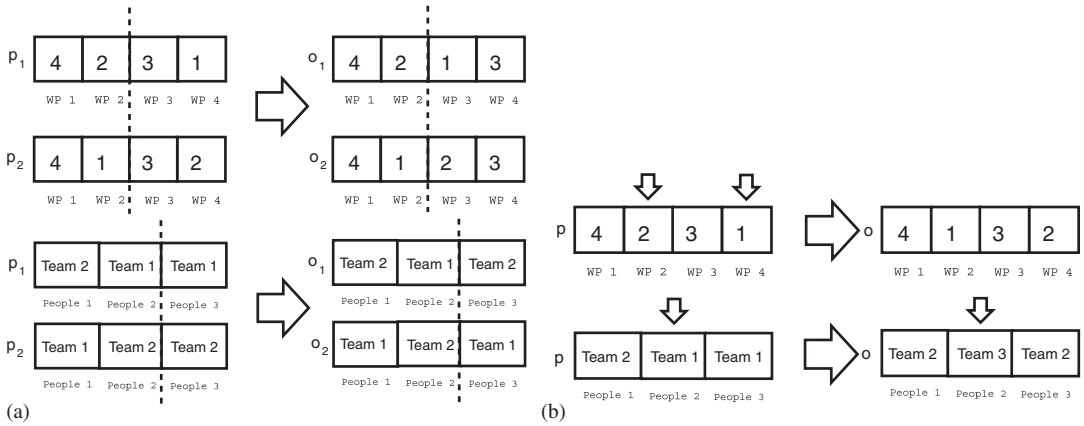

Figure 6. Crossover and mutation operators for WP ordering and developers allocation to teams: (a) crossover and (b) mutation.

between the fitness values of the two neighbor individuals being compared. The effect of 'cooling' on the simulation of annealing is that the probability of following an unfavorable move is reduced. This (initially) allows the search to move away from the local optima in which the search might be trapped. As the simulation 'cools' the search becomes more and more like a simple hill climb.

Below we detail the operators used for the GA. SHC and SA can be applied using, as operator for generating a neighbor solution from an existing one, the mutation operator defined for the GA.

4.1.1. GA configuration and operators. To construct a GA for a given problem, we need to define the representation, the fitness function, the GA operators (selection, crossover, and mutation) and choices of settings for other GA parameters. We used a simple (i.e. with a non-overlapping population) GA, with a *roulette wheel* selection mechanism and an elitism of two individuals (i.e. the best two individuals are guaranteed to survive across generations). The GA representation adopted is the two-array representation of Figure 2, extended to the n-array representation when separate queues with teams of specialized developers are needed (Figure 5(b)). The fitness function is evaluated as described in Section 3.4. The population is randomly initialized with individuals where (i) the WP ordering is randomly determined and (ii) for each queuing node the available staff is randomly grouped into teams. The crossover operator (Figure 6(a)) produces an offspring of two new individuals (o_1 and o_2) starting from two parents (p_1 and p_2). The first row of the parents (representing the WP ordering) are re-combined as follows:

1. A random position k, is selected in the chromosome.
2. The first k elements of p_1 become the first k elements of o_1.
3. The last $N-k$ elements of o_1 are the sequence of $N-k$ elements which remain when the k elements selected from p_1 are removed from p_2.
4. o_2 is obtained similarly, composed of the first $N-k$ elements of p_2 and the remaining elements of p_1 (when the first $N-k$ elements of p_2 are removed).

For the remaining rows (encoding the staff distribution to each queue handling requests for a particular expertise), a standard *single-point* crossover is used.

The mutation operator (Figure 6(b)) works as follows:

- It randomly decides whether to alter the WP ordering or the staff distribution.
- If the WP ordering is to be mutated then the mutation operator randomly selects two WPs (i.e. two array items) and exchanges their position in the queue.
- If the staff distribution is to be mutated, then
  - if there are separate queues, the queuing node (i.e. the chromosome row r—where $2 \leq r \leq \|Ex\|+1$, $\|Ex\|$ is the number of separate queues handling requests for particular expertises—in which the mutation has to be performed is randomly chosen;

DOI: 10.1002/spe

- mutation is applied, by randomly choosing a staff member and changing his/her team. It is worth pointing out that the random number generated by the mutation operator is an integer ranging from 1 to the maximum number of staff available. The cardinality of the set of numbers present in the genome determines the number of teams (servers) of the queuing model. Clearly, a cardinality equal to the number of developers available indicates that each team is composed of a single staff member, whereas a cardinality equal to one indicates that a single server (composed of all staff) is used.

After a new individual has been generated by means of the initialization, crossover or mutation operator, it is repaired as discussed in Section 3.4, in order to ensure that the precedence constraints are respected.

4.1.2. Configuration and operators for the multi-objective GA. When a multi-objective optimization, as mentioned in Section 3.4.1, needs to be performed, we used the Non-dominated Sorting Genetic Algorithm II (NSGA-II) approach proposed by Deb *et al.* [48]. NSGA-II is a multi-objective optimization algorithm that incorporates elitism to maintain the solutions of the best front found.

A naive multi-objective optimization algorithm would require $\mathcal{O}(M\ N)$ comparisons to identify each solution of the first non-dominated front in a population of size N and with M objectives, and a total of $\mathcal{O}(M\ N^2)$ comparisons to build the first non-dominated front. This is because each solution needs to be compared with all other solutions. As the above step has to be repeated for all possible fronts—which can be at most N, if each front is composed of one solution—the overall complexity for building all fronts is $\mathcal{O}(M\ N^3)$.

NSGA-II uses a faster algorithm for non-dominated sorting, which has a complexity $\mathcal{O}(M\ N^2)$ (as shown in the paper by Deb *et al.* [48]):

1. For each solution p in the population, the algorithm finds the set of solutions S_p dominated by p and the number of solutions n_p that dominate p. The set of solutions with $n_p = 0$ are placed in the set first front F_1.
2. $\forall p \in F_1$, solutions $q \in S_p$ are visited and, if $n_q - 1 = 0$, then solution q is placed in the second front F_2. This step is repeated $\forall p \in F_1$ to generate F_3, etc.

To compare solutions, NSGA-II uses the 'crowded comparison operator'. That is, given two solutions x_1 and x_2, x_1 is preferred over x_2 if it belongs to a different (better) front. Otherwise, if x_1 and x_2 belong to the same front, the solution located in the less crowded region of the front is preferred.

Then, NSGA-II produces the generation $t+1$ from generation t as follows:

1. generating the child population Q_t from the parent population P_t using the binary tournament selection and the crossover and mutation operators defined for the specific problem;
2. creating a set of $2N$ solutions $R_t \equiv P_t \cup Q_t$;
3. sorting R_t using the non-domination mechanism above described, and forming the new population P_{t+1} by selecting the N best solutions using the crowded comparison operator.

For the particular problem of balancing between the completion time and the staffing level, crossover and mutation operators need to be slightly changed to allow for a variable staffing level. The problem is simply dealt by using, for the developer allocation to teams, the same pigeon-hole representation described in Section 3.3, allocating as many slots in the array as the maximum number of developers available for the queue. Then, each slot can either contain a non-negative integer indicating the team where the developer is allocated, or -1 if the developer is not allocated at all. In such a case:

1. The crossover operator works as described in Section 4.1.1. However, in case one individual of the produced offspring exhibits a staffing allocation below a (fixed) lower bound, then the crossover is rolled back.

DOI: 10.1002/spe

2. The mutation operator randomly selects a developer and assigns it to a randomly chosen team, or to the pool of unassigned developers (slot value $= -1$). Also in this case, if the total staffing of the queue is below a lower bound, the mutation is rolled back.

5. EMPIRICAL STUDY

The objective of this empirical study is to evaluate the effectiveness and the performance of different search-based techniques, combined with queuing simulation, to determine project planning, i.e. the distribution of the staffing across teams and WPs assignment to teams.

The context of our study constitutes two maintenance projects, hereby referred as *Project A* and *Project B*. *Project A* was a massive maintenance project aimed at fixing the Y2K problem in a large financial software system from a European financial organization. The entire system was decomposed into 84 WPs, each one composed, on average, of 300 COBOL and JCL files. No WP dependence was documented and thus no constraint has to be satisfied in *Project A* scheduling. Further details can be found in [3].

Project B is composed of 108 WPs for which WP inter-dependence information is available. Though a little smaller (in terms of total effort required) than *Project A*, the presence of a total of 102 dependences between the project's WPs (Figure 7) considerably complicates the problem of project management. The project is aimed at delivering the next release of a large data intensive software system, written in several languages, including DB II, SQL, and .NET™. The project involved members of staff who had been developing the project previously and hence unfamiliarity with the work and consequent training time was not an issue. Although all developers taking part in the project worked on any WP, it is possible to classify WPs according to the main expertise required for each one. In particular, there are 32 WPs related to database creation and maintenance, 14 to Graphical User Interface, 11 related to networking (LDAP), 12 related to Middleware, and 16 related to the core application domain.

This information will be used in Section 5.3.4 to simulate the possibility of employing specialized developers, i.e. developers having the expertise required to process a given WP (in this project, each WP requires only one expertise). Descriptive statistics of WP efforts for the two projects are reported in Table I.

Although both *Project A* and *Project B* are maintenance projects, this does not lead the applicability of the proposed approach to other activities, such as coding and testing.

5.1. Research questions

The research questions this study aims at investigating are the following:

RQ1: Performance
How do different search algorithms compare in terms of project completion time after a fixed number of evaluations?
RQ2: Effect of varying the staffing level
How do estimated project completion times and the distribution of developers across teams vary under different staffing levels, i.e. total number of developers available?
RQ3: Fragmentation effects
What is the effect of staffing level and completion time on fragmentation?
RQ4: Employment of specialized developers
What will be the completion time for different staffing levels if WPs need to be dispatched to teams of specialized developers?

5.2. Empirical study settings and instrumentation

In order to facilitate replication of our work, in this section we report the details of the parameter settings used in our experiments. Settings were performed by means of a trial-and-error procedure, starting from parameter settings suggested by the literature [49].

DOI: 10.1002/spe

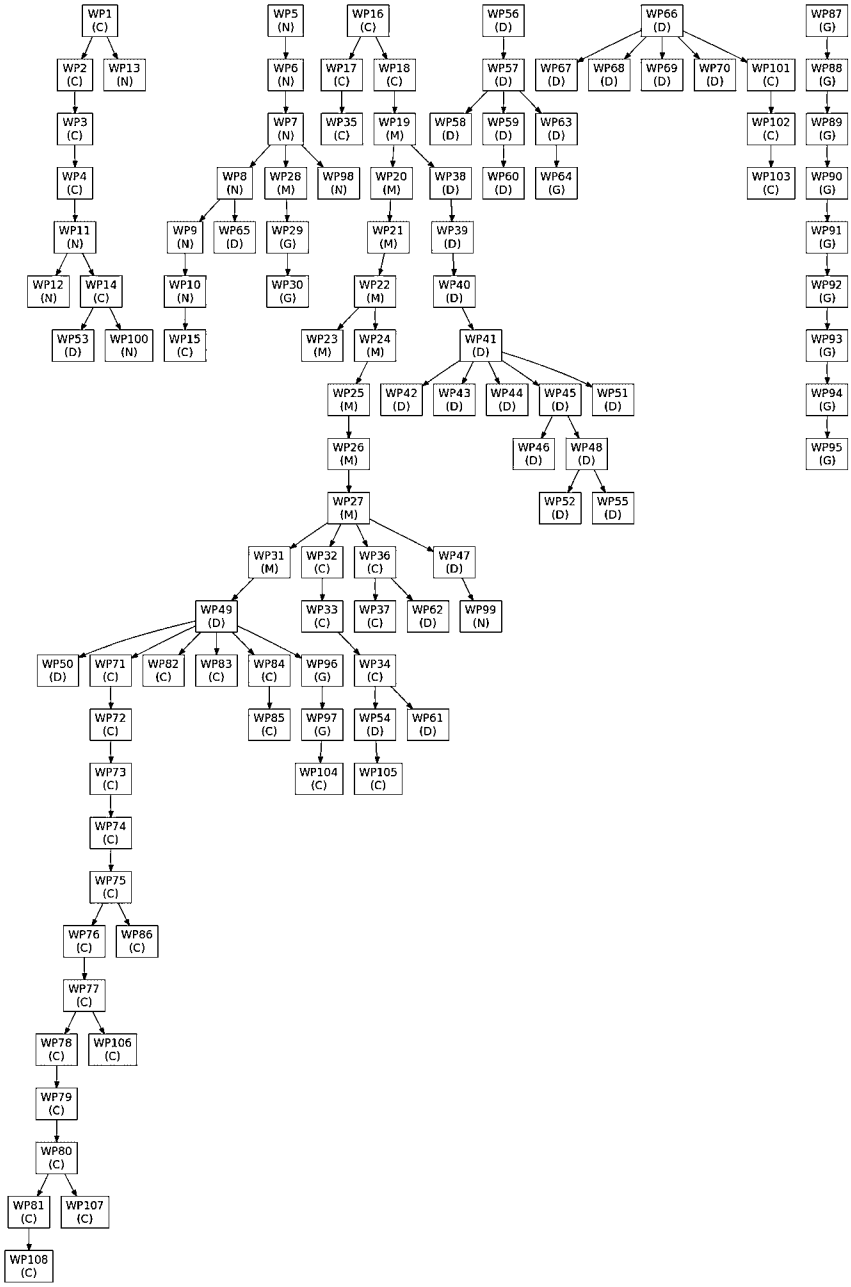

Figure 7. Dependences among WPs for *Project B*. Letters indicate the WP category (C, core; D, database; N, networking; M, middleware; G, GUI).

SA parameters were set up as follows: maximum temperature $T_{\max} = 0.30$, minimum temperature $T_{\min} = 0.01$, cooling factor $r = 0.001$. GA simulations were run considering the following parameters:

1. non-overlapping GA with elitism of two individuals;
2. population composed of 50 individuals;
3. 250 generations for *Project A* and 100 for *Project B*. Such a stopping criterion was determined by identifying the number of generations over which no further improvement was obtained (doubling the number of generations we obtained an improvement of less than

DOI: 10.1002/spe

Table I. Descriptive statistics for the two projects (effort is expressed in person days).

	Project A	Project B
Total no. of WPs	84	108
Dependences	—	102
Min. effort	2	1
Max. effort	306	60
Median effort	29	4
Mean effort	51	5
Std. dev. effort	59	6
Total effort	4287	525

5%). The number of generations for *Project B* is smaller since, because of the presence of dependences, the degrees of freedom for WP ordering are reduced;

4. mutation probability 0.1, crossover probability 0.7.

Multi-objective optimization (used to address research questions 3 and 4) was performed with an NSGA-II algorithm for which the same configuration used for the single-objective GA was used. To make the comparison between different algorithms fair, the number of restarts for SHC and SA was chosen so as to generate the same number of solutions as for GA, and in any case the algorithm is stopped once such a maximum number of evaluations has been reached.

To reduce the bias of randomness, each experiment was repeated 30 times and the statistics of all runs are reported in the form of boxplots.

5.3. Empirical study results

This subsection reports results from the application of the proposed search-based staffing approach to the two projects *Project A* and *Project B*.

5.3.1. Comparing the different search algorithms. To answer *RQ1* and *RQ2*, we analyzed the evolution of the minimum completion time over the same number of solutions generated by the different algorithms and by random search. In this section we aim at minimizing completion time only (without considering fragmentation) intended to be one of the ultimate goals for a project manager, which requires to complete the project in the minimum time possible, or to meet a deadline negotiated with the customer.

For GA this means that it generates a number of solutions $= population\ size \cdot number\ of\ generations$. Random search generates each time a random WP ordering and a random assignment of developers across teams. Comparisons were performed for a fixed staffing level, i.e. 46 developers for *Project A* (as estimated in a previous work [3]), and 20 developers (the actual staffing) for *Project B*. Figure 8 shows the boxplots of estimates obtained by the different algorithms for 30 runs.

First, we compared every algorithm with random search and with other algorithms, using the non-parametric two-tailed Mann–Whitney test. The results shown in Table II and the boxplots in Figure 8 indicate that all algorithms outperform random search. Moreover, for both projects SA and GA outperformed SHC. However, while in *Project A*—the project without dependences between WPs—SA significantly outperformed GA, this is not the case for *Project B*—the project with dependences between WPs—where no significant difference was found between SA and GA. In the following, to answer the remaining research questions, we report the results for SA in the case of *Project A* and for GA in the case of *Project B*.

5.3.2. Completion time and distribution of developers for different staffing levels. **RQ2** analyzes the effect of staffing levels on completion time and on distribution of developers across teams. Such a research question is relevant for a manager who would like to decide how to adjust his/her project staffing to be able to complete a project within a given date. Figure 9 shows boxplots of

DOI: 10.1002/spe

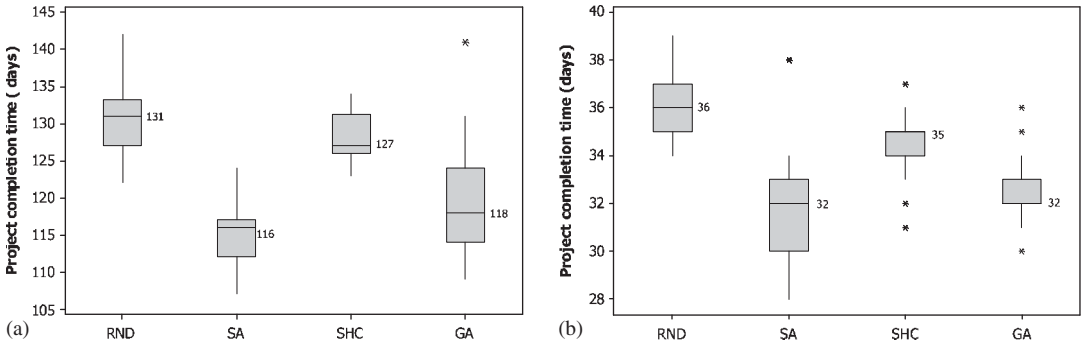

Figure 8. Boxplots of estimated project completion times for different algorithms: (a) Project A and (b) Project B.

Table II. Comparison between different algorithms using a two-tailed Mann–Whitney unpaired test of estimated project completion times.

Project		SA	SHC	GA
A	RND	3×10^{-11}	0.015	3×10^{-11}
	SA	—	4×10^{-11}	0.015
	SHC	—	—	3×10^{-11}
B	RND	2×10^{-7}	2×10^{-6}	1×10^{-10}
	SA	—	2.9×10^{-5}	0.18
	SHC	—	—	1×10^{-5}

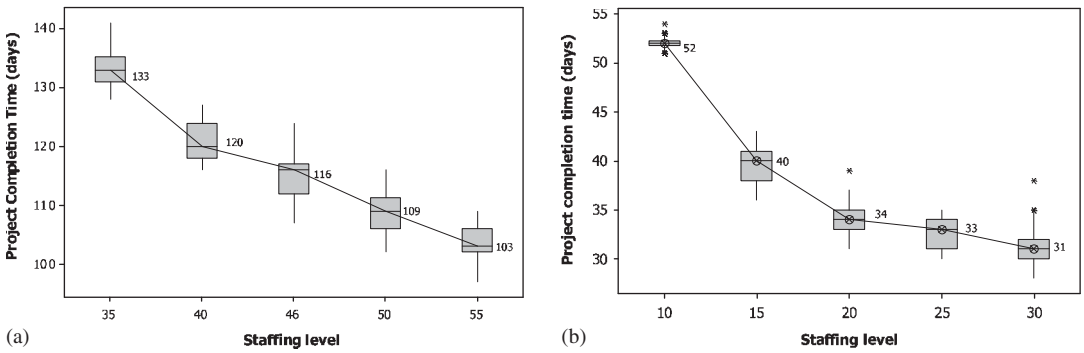

Figure 9. Boxplots of estimated project completion times for different staffing levels: (a) Project A and (b) Project B.

the estimated completion times for the two projects. To evaluate the cost/benefit trade-off due to staff increasing, we used the ratio $\Delta time/\Delta staffing$. The results are different for the two projects. For *Project A* the staffing increase 35–40 exhibit a high ratio (2.6); then for further increases the ratio decreases (0.66 for the increase 40–46, 1.7 for 46–50, and 1.2 for 50–55). For *Project B* the ratio is high (2.4) for the increase 10–15, then it decreases to 1.2 for 15–20. For further staffing increases, the ratio tends to be very small (0.2 for 20–25 and 0.4 for 25–30).

Let us now analyze how developers are distributed across teams. Figure 10 shows the number of allocated teams for different staffing levels. For *Project A* the median number of teams tends to increase only slightly for staffing levels above 50. For *Project B*, where there are WP dependences,

DOI: 10.1002/spe

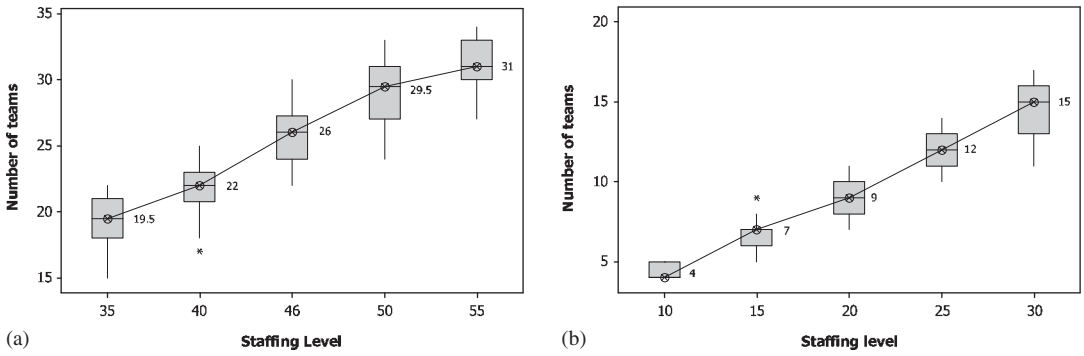

Figure 10. Boxplots of number of allocated teams for different staffing levels: (a) Project A and (b) Project B.

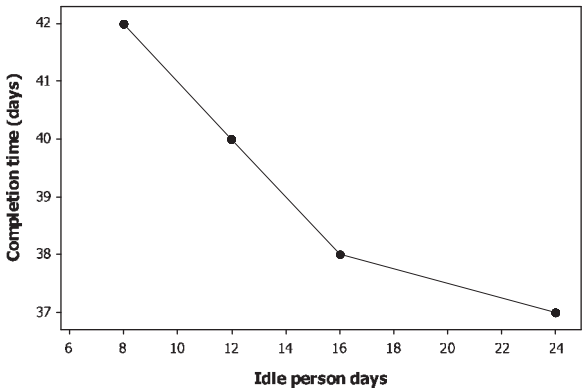

Figure 11. Project B: Trade-off between completion time and schedule fragmentation.

a staffing increase between 25 and 30 does not reduce the completion time, but causes an increase in the number of allocated teams.

5.3.3. Reducing schedule fragmentation. **RQ3** deals with reducing schedule fragmentation, which can occur in all cases where there are dependences among WPs. In fact, teams might have to wait until blocking dependences of an incoming WP are solved (as in *Project B*). Figure 11 shows an example of Pareto front obtained by means of a multi-objective NSGA-II optimization, as explained in Section 3.4.1, and considering a staffing level of 20 developers. The two conflicting objectives are the completion time and the idle person days. Using the Pareto front, the manager can select solution achieving, for example a low number of idle person days (8), accepting a higher completion time (42 days). Alternatively, the manager can see that there also exist valid solutions with a higher number of idle person days (24) but a shorter completion time (37).

In the Pareto-based search, all objectives are treated equally so that no trading off between objectives takes place until the manager consults the Pareto front. In this way, the Pareto-based approach makes a suitable delineation of responsibilities, separating those aspects of the problem that require human expertise and judgment from that which can be easily automated and which concern the tiresome, repetitive, and less imaginative aspects of the search.

5.3.4. Dispatching WPs to specialized teams. To answer **RQ4**, we simulated, for *Project B*, the availability of developers having specialized skills. A multi-objective NSGA-II algorithm was used to determine WP allocation and developers distribution across specialized teams. In this case, the NSGA-II considered as objectives the project completion time and the staffing level. As in this case we had to deal with expertise, the model was instantiated as explained in Section 3.5 and shown in

DOI: 10.1002/spe

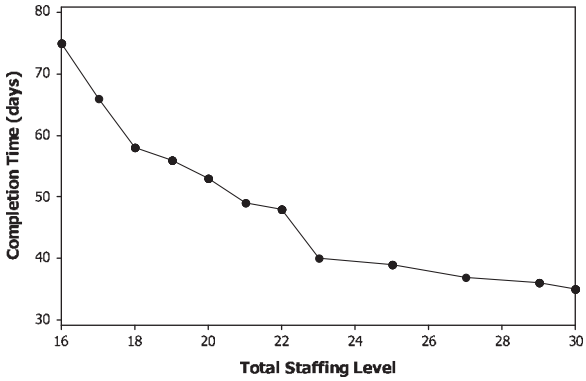

Figure 12. Dispatching requests to specialized teams.

Table III. Staffing of specialized teams: team composition for different staffing levels.

Completion time	Total staffing	Database	GUI	Networking	Middleware	Core
58	18	1,2	1,1,2	1,1	1,2	1,2,3
56	19	1,2	1,1,2	1,1,1	1,2	1,2,3
53	20	1,1,2	1,1,1	1,1,1,1	3	1,2,3
49	21	1,2	1,1,1,1	1,1,1	1,2	1,3,4
48	22	1,2	1,1,1	1,1,1,1	1,3	1,3,4
40	23	2,2	1,1,1	1,1,2	1,3	1,3,4
39	25	2,2	1,1,2	1,1,3	1,3	1,3,4
37	27	2,2	1,1,1,1	1,1,3	1,3	2,3,5

Figure 5, i.e. with five separate queue, each one handling WP requiring a particular expertise. The algorithm considered solutions with a staffing level, for each queue, varying between 2 and 10, which resulted in an overall staffing level between $2 \times 5 = 10$ and $10 \times 5 = 50$ developers.

The multi-objective NSGA-II produced Pareto fronts composed of solutions for different staffing levels and different completion times. An example is shown in Figure 12. As expected, completion times tend to be higher than the ones with the same staffing level reported in Figure 9(b), where it was assumed that developers could have worked on any WP. The inclusion of skills-match as a constraint makes the problem harder with a tendency to increase completion time. For example, for a staffing level of 20 developers the completion time increased from 34 to 50; for higher staffing levels (30) the difference is smaller, i.e. from 31 to 34.

Table III shows how resources are allocated across teams for different staffing levels. Note that the configurations reported here only represent examples of solutions that can be obtained with those staffing levels. x, y, z in a column means that the algorithm allocated, for that queue, three teams composed of x, y, and z developers, respectively. The search process yields insight into the effects of skills-matching on the team sizes. The search is guided by fitness and hence there is a 'logic' to the solutions that it is able to locate in the solution space. Inspection of the results reveals some interesting relationships between the structure of the dependences in the project and the way in which the search-based algorithms seek to optimize the allocation of staff to teams.

For example, consider the Database queue. For this queue, the algorithm attempts to create teams composed of two developers. Database WPs constitute the majority of the WPs, thus they required short maintenance times, although in most cases they did not block other WPs, as shown in Figure 7. Perhaps the most interesting situation occurs with the GUI queue: the number of developers did not necessarily increase with a higher total staffing (and a smaller completion time). This is because almost all GUI WPs are independent of other WPs. Therefore, the project completion time tends to depend upon other WPs.

DOI: 10.1002/spe

Notice that, for the Network queue, the algorithm allocated many singleton teams. This happens because Networking WPs impose dependences on several groups of other WPs. Therefore, working on Networking tasks in parallel is highly efficient for this problem structure. For Middleware, the algorithm allocated a larger team (composed of two to four developers) plus (except for a staffing of 20) one singleton team. This is because most of these WPs were sequentially dependent: the only way to complete them in a short time is to create larger teams. Finally, the team allocation was consistent, when increasing the staffing, for the Core queue: a smaller team, a medium team, and a larger team to be able to handle Core WPs that were of different sizes and on which many other WPs are dependent.

5.4. Discussion

When comparing different algorithms (**RQ1**) it is evident that SA and GA outperform SHC. Despite the use of random restarting, hill climbing still suffers from the fact that solutions tend to get trapped into local optima, while it is known that GA can avoid this problem and that SA is able to improve it because of its capacity to temporarily accept worse solutions. The performance of both GA and SA in our experiments provide evidence to support the claim that it is worthwhile adopting one of these two algorithms in a tool that act as the project management assistant.

The proposed approach permits the decision maker to analyze the variation in the best completion time when increasing or decreasing the staffing level (**RQ2**). As might be expected, the case studies reveal that, in the presence of a large number of dependences, staffing increments over certain values do not produce improvements in project completion time, since dependences limit the level of parallelism that can be achieved. Of course, in both cases we are considering projects where Brooks' law does not apply, thus there is a linear relationship between staffing levels and the time needed to perform the task. Once again, the availability of analyses performed with the proposed approach gives the manager the ability to analyze the return on investment of staff time in terms of reduced completion time. This may help to facilitate the negotiation of deadlines and costs with the software customer.

Differences between the two projects studied in this paper can be seen by considering how staff is distributed across teams when varying the staffing level. Where there are no dependences, the approach tends to parallelize the work as much as possible, by creating a large number of small teams. In projects with a high number of dependences—such as *Project B*—larger teams appear more desirable, since they can be used to quickly process blocked tasks. On the other hand, this is also true when the project contains very large WPs: such is the case with *Project A*, where, above a certain staffing level, the search tends to locate solutions that increase the size of existing teams rather than creating new ones.

When handling projects with dependences among WPs, such as *Project B*, one phenomenon that should be limited as much as possible is schedule fragmentation (**RQ3**). Clearly, there may be cases in which fragmentation might be acceptable, e.g. when developers are able to work on different projects at the same time or when there is the possibility of using the idle time for other activities such as training. By using multi-objective NSGA-II optimization, the manager can choose among a Pareto front of solutions achieving different compromises between project completion time and idle person days.

When using queues with teams composed of specialized developers (**RQ4**), it was noted that the algorithm tends to understaff queues for categories of WPs that do not block the entire project (e.g. GUI-related WPs for *Project B*), whereas it creates a small number of larger teams if WPs of a particular category (e.g. Middleware) must be worked sequentially. In other cases the algorithm tends to balance between having many smaller teams working in parallel with large teams able to quickly complete a WP.

Finally, it is interesting to compare the results of our approach with the real project staffing level and people organization. While for *Project A* this information was fully available, for *Project B* (which comes from a different company) it was not possible to include real data about the projects' actual completion time, but only about the staffing level and the distribution of developers across teams.

DOI: 10.1002/spe

Table IV. Project B: Comparison between generated and actual team allocation.

Config.	Singleton allowed	No. of teams for different sizes							Total no. of teams
		1	2	3	4	5	6	7	
Generated	Yes	3	1	5	0	0	0	0	9
Generated	No	0	4	1	1	1	0	0	7
Actual	—	0	1	2	0	1	0	1	5

For *Project A* the actual staffing level was roughly 80 developers whereas the completion time was 155 working days. The number of teams varied during the time between 2 and 27 with a median value of 6, as shown in previous papers [3, 50]. The number of teams working concurrently on a given day (thus the number of servants in a queuing model) varied from 1 to 12, with a median value of 6. This confirms the fact that the industry managers avoided having a large number of teams composed of few developers, preferring, instead, few teams composed of more developers. This permitted to minimize the risks, also because developers were also working on other, different tasks. If comparing, for the same staffing level (46 developers) the completion time with the one estimated in [3], the new developers distribution and WP assignment to team permitted a median completion time of 116 working days instead of 155 working days.

Different from *Project A*, the resource allocation for *Project B* was done at the beginning of the project and not changed anymore. Table IV compares the actual distribution of developers across teams with the ones instantiated by our approach. As shown, the actual assignment is closer to the automatically generated assignment when preventing the creation of singleton groups. This confirms the conjecture that managers tend not to create such a kind of teams. The most tangible difference is the presence in the actual allocation of a team composed of seven developers, supplied, in the generated allocation, by a large (four) number of two-developers teams, that can better permit the parallelization of work.

5.5. *Limitations, assumptions, and threats to validity*

The evaluation of the approach presented in this paper relies on the two real-world projects for which we were able to obtain data. Naturally, these projects may not be typical and hence care is required in extrapolating the results. In particular, neither project had mutually interdependent WPs (cyclical dependences) or WPs that had to be processed in parallel at the same time nor requirements for specific skill assignments. Naturally, this may be different in other software projects, thereby requiring a change in formulation. Notwithstanding the precedence constraints imposed by dependences, non-overlapping WPs mean that each WP can be treated as an atomic unit of work. A different formulation would be required to handle overlapping WPs.

Expertise was treated as Boolean property; either a developer has the expertise or he does not. Our real-world projects did not have expertise data and we did not think it right to develop sophisticated approaches based on mere speculation. Finally, we also assumed that each WP required only a single expertise.

Each of the two projects also has its own set of limitations and assumptions, that were made by the managers who collected the data and this has a consequent effect upon the way in which we are able to experiment with the data.

Project A was the simpler of our two real-world projects, because it involved maintenance interventions performed almost semi-automatically and involved highly standardized activities. The project involved searching for year fields and making them Y2K compliant by inserting a source code fragment implementing a windowing mechanism. In such conditions, it is possible to make the assumption of interchangeability between people and months. That is, given a maintenance team size, s and the effort required e, the time t necessary to perform the task is $t = e/s$.

Of course, in general, due to Brooks' law [40], this could be an overly optimistic assumption. However, as other authors have noted [39], given the small team sizes (fewer than eight developers) and the standard (training-free) nature of the maintenance task, this approximation was considered

DOI: 10.1002/spe

reasonable. Therefore, the results may not be limited merely to similar 'defined task' maintenance interventions but may also apply to other projects.

For *Project B*, we also make the assumption that staff can be allocated to different WPs without the need for re-training. This does not directly contravene Brooks' law, because we make no attempt to allocate new staff to work on a WP that is already underway. Thus, it could be said that we observe Brooks' law at the micro level, but at the macro level, we assume that there is no need for re-training.

The remainder of this section, formally considers the threats to validity that may affect the extent to which the empirical results of any study can be generalized. We shall consider these general treats to validity and the way in which they affect the extent to which the results presented in the paper can be generalized and relied upon.

Construct validity threats may be due to the simplifications made on the maintenance process modeled, as well as to the assumptions made in Section 5.5. However (i) more complex maintenance processes can be modeled using queuing networks as shown in paper [3] without preventing the applicability of the proposed approach, (ii) this paper has shown how it is possible to consider a multi-queue configuration to allocate teams of specialized developers, and (iii) effects, such as communication overhead, can be introduced in the relationship between completion time and effort as shown in [41].

Internal validity threats, in our case study, can be due to the randomness of the results obtained when running the different algorithms. To limit such a threat, different actions were taken:

- First and foremost, we carefully calibrated the parameters for each algorithm, the number of comparisons needed by SA, SHC, and RND and the number of generations and the population size needed by GA. The chosen values were determined ensuring that further increases did not significantly affect the results, and we compared the different algorithms over the same number of solutions generated.
- To avoid the results being affected by randomness, as mentioned in Section 5.2 each experiment was performed 30 times, and the statistics over the different runs were reported.

As regards *External validity*, as explained earlier, the results obtained can be extended to projects where communication overhead does not affect the performance of the teams. For all the other cases, there is the need for a more complex model. Nevertheless, this does not affect the applicability of the overall approach, once factors such as communication overhead have been modeled in the fitness function, as shown in other work [41].

6. CONCLUSION

This paper presented a comprehensive treatment of a search-based approach to software project planning, showing how search-based techniques can be used to address problems of staffing level adjustment, allocation of staff to teams, reduction of project fragmentation and team composition based on different programmer expertise and WP required knowledge. These problems are believed to be particularly acute in software projects, yet hitherto, there has been little work on the application of search-based optimization techniques for addressing these problems.

The paper reported empirical results from the application of search techniques to two large-scale, real-world maintenance and evolution projects. The results show that both SA and GAs can construct solutions that have the potential to provide valuable decision support to software managers. The results also show that the search-based approach implicitly maximizes parallelism in the project to shorten completion time and reduce fragmentation. The data also shed light on the impact of dependences in software projects. Finally, the approach allows for balancing conflicting objectives such as completion time, staffing level, and resource allocation by using multi-objective optimization, providing the manager with Pareto fronts of possible solutions rather than single solutions.

DOI: 10.1002/spe

The proposed approach cannot be considered a substitute to the managers' activity related to project planning and staffing: personal experience, plus the knowledge of the organizational structure, of people skills and working attitude are vital to effectively plan a project. The manager is the sole judge of feasibility and the soundness of computed solutions; approaches as the one presented in this paper aim at supporting manager activity when the space of solutions is so large that the manual or exhaustive search become daunting tasks. Indeed, (semi)automatic approaches can be a useful contribution, providing managers with initial solutions that can be refined taking into account factors not captured by the model.

Work-in-progress includes empirical studies aimed at evaluating the robustness and sensitiveness of schedules generated by the proposed approach [31]. Also the future work is devoted to adding further levels of complexity to the proposed model, for example:

- the presence of interdependent (overlapping) WPs requiring some joint work;
- the use of a more sophisticated model of expertise. For instance, instead of considering expertise as a Boolean property, one can imagine that there might be developers with low, medium or high expertise, as in the work of Gutjahr *et al.* [35] and Stummer *et al.* [36]. Moreover, complex projects might require more than one expertise for each WP, thus future extensions of the approach proposed in this paper should support that.
- the possibility of periodically re-organizing the people distribution to teams, as well as the possibility (as in [35, 36]) that developers might only spend part of their available time on the WP they have been assigned to.

Finally, we intend to consider further optimization techniques, in particular to handle multi-objective optimization (e.g. comparing solutions provided by NSGA-II with the exact Pareto front).

ACKNOWLEDGEMENTS

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (Research Chair in Software Evolution #950-202658) and by G. Antoniol Individual Discovery Grant. Mark Harman is supported by EPSRC project Software Engineering by Automated Search (EP/D050863, SEBASE, 2006-2011), for which the other principal investigators are John Clark (University of York) and Xin Yao (University of Birmingham) and by an EPSRC platform grant (EP/G060525, CREST, 2009-2014).

REFERENCES

1. Boehm BW, Horowitz E, Madachy R, Reifer D, Clark BK, Steece B, Winsor Brown A, Chulani S, Abts C. *Software Cost Estimation with COCOMO II*. Prentice-Hall: Englewood Cliffs, NJ, 2000.
2. Shepperd MJ, Schofield C. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering* 1997; **23**(11):736–743.
3. Antoniol G, Cimitile A, Di Lucca GA, Di Penta M. Assessing staffing needs for a software maintenance project through queuing simulation. *IEEE Transactions on Software Engineering* 2004; **30**(1):43–58.
4. Garey MR, Johnson DS. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman WH: New York, 1979.
5. Husbands P. Genetic algorithms for scheduling. *AISB Quarterly* 1994; **89**:38–45.
6. Deb K. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley: New York, NY, 2001.
7. Davis L. Job shop scheduling with genetic algorithms. *Proceedings of the 1st International Conference on Genetic Algorithms*. L. Erlbaum Associates Inc.: Hillsdale, NJ, U.S.A., 1985; 136–140.
8. Coffman EG, Garey MR, Johnson DS. Approximation algorithms for bin packing: A survey. *Approximation Algorithms for Np-Hard Problems*, Hochbaum DS (ed.). PWS Publishing Co.: Boston, MA, 1997; 46–93.
9. Hart E, Corne D, Ross P. The state of the art in evolutionary scheduling. *Genetic Programming and Evolvable Machines* 2004.
10. Kolisch R, Hartmann S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research* 2006; **174**(1):23–37.
11. Harman M. The current state and future of search based software engineering. *Future of Software Engineering 2007*, Briand L, Wolf A (eds.). IEEE Computer Society Press: Los Alamitos, CA, U.S.A., 2007.
12. Clark J, Dolado JJ, Harman M, Hierons RM, Jones B, Lumkin M, Mitchell B, Mancoridis S, Rees K, Roper M, Shepperd M. Reformulating software engineering as a search problem. *IEE Proceedings—Software* 2003; **150**(3):161–175.

DOI: 10.1002/spe

13. Harman M, Mansouri A, Zhang Y. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Technical Report TR-09-03*, Department of Computer Science, King's College London, 2009.
14. Chao C, Komada J, Liu Q, Muteja M, Alsalqan Y, Chang C. An application of genetic algorithms to software project management. *Proceedings of the Ninth International Advanced Science and Technology*, Chicago, IL, U.S.A., 1993; 247–252.
15. Chang CK, Chao C, Hsieh S, Alsalqan Y. SPMNet: A formal methodology for software management. *Proceedings of the 18th Annual International Computer Software and Applications Conference* (*COMPSAC '94*), Taipei, Taiwan, 9–11 November. IEEE: New York, 1994; 57–57.
16. Chang CK, Chao C, Nguyen TT, Christensen M. Software Project Management Net: A New Methodology on Software Management. *Proceedings of the 22nd Annual International Computer Software and Applications Conference* (*COMPSAC '98*), Vienna, Austria, 19–21 August. IEEE Computer Society Press: Silver Spring, MD, 1998; 534–539.
17. Chang CK, Christensen MJ, Zhang T. Genetic algorithms for project management. *Annals of Software Engineering* 2001; **11**(1):107–139.
18. Aguilar-Ruiz JS, Ramos I, Riquelme Santos JC, Toro M. An evolutionary approach to estimating software development projects. *Information and Software Technology* 2001; **43**(14):875–882.
19. Aguilar-Ruiz JS, Riquelme Santos JC, Ramos I. Natural evolutionary coding: An application to estimating software development projects. *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation* (*GECCO '02*), New York, U.S.A., 9–13 July 2002; 1–8.
20. Alba E, Chicano F. Software project management with GAs. *Information Sciences* 2007; **177**(11):2380–2401.
21. Alba E, Chicano F. Management of software projects with GAs. *Proceedings of the Sixth Metaheuristics International Conference* (*MIC '05*), Vienna, Austria. Elsevier: Amsterdam, 2005; 13–18.
22. Antoniol G, Di Penta M, Harman M. Search-based techniques applied to optimization of project planning for a massive maintenance project. *21st IEEE International Conference on Software Maintenance*, Los Alamitos, CA, U.S.A. IEEE Computer Society Press: Silver Spring, MD, 2005; 240–249.
23. Alba E, Luque G, Luna F. Parallel metaheuristics for workforce planning. *Journal of Mathematical Modelling and Algorithms* 2007; **6**(3):509–528.
24. Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F. A scatter search algorithm for project scheduling under partially renewable resources. *Journal of Heuristics* 2006; **12**(1–2):95–113.
25. Barreto A, de O. Barros M, Werner C. Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers and Operations Research* 2008; **35**(10):3073–3089.
26. Cortellessa V, Marinelli F, Potena P. An optimization framework for 'Build-Or-Buy' decisions in software architecture. *Computers and Operations Research* 2008; **35**(10):3090–3106.
27. Kapur P, Ngo-The A, Ruhe G, Smith A. Optimized staffing for product releases and its application at Chartwell Technology. *Journal of Software Maintenance and Evolution*: *Research and Practice* (*Special Issue Search Based Software Engineering*) 2008; **20**(5):365–386.
28. Dolado JJ. On the problem of the software cost function. *Information and Software Technology* 2001; **43**(1):61–72.
29. Kirsopp C, Shepperd M, Hart J. Search heuristics, case-based reasoning and software project effort prediction. *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation* (*GECCO '02*), New York. Morgan Kaufmann: Los Altos, CA, 9–13 July 2002; 1367–1374.
30. Shepperd M. Software economics. *Future of Software Engineering 2007*, Briand L, Wolf A (eds.). IEEE Computer Society Press: Los Alamitos, CA, U.S.A., 2007.
31. Gueorguiev S, Harman M, Antoniol G. Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. *Genetic and Evolutionary Computation Conference*, *GECCO 2009*, *Proceedings*, Montreal, QC, Canada. ACM: New York, July 8–12 2009; 1673–1680.
32. Ramaswamy R. How to staff business critical maintenance projects. *IEEE Software* 2000; **7**(7):90–95.
33. Podnar I, Mikac B. Software maintenance process analysis using discrete-event simulation. *European Conference on Software Maintenance and Reengineering*, Lisbon, Portugal. IEEE Society Press: Silver Spring, MD, March 2001; 192–195.
34. Bertolino A, Marchetti E, Mirandola R. Performance measures for supporting project manager decisions. *Software Process*: *Improvement and Practice* 2007; **12**(2):141–164.
35. Gutjahr W, Katzensteiner S, Reiter P, Stummer C, Denk M. Competence-driven project portfolio selection, scheduling and staff assignment. *Central European Journal of Operations Research* 2008; **16**(3): 281–306.
36. Stummer C, Kiesling E, Gutjahr WJ. A multicriteria decision support system for competence-driven project portfolio selection. *International Journal of Information Technology and Decision Making* 2009; **8**(2):379–401.
37. Barreto A, de Oliveira Barros M, Werner C. Staffing a software project: A constraint satisfaction approach. *ACM SIGSOFT Software Engineering Notes* 2005; **30**(4):1–5.
38. Raffo DM, Kellner MI. Empirical analysis in software process simulation modeling. *Journal of Systems and Software* 2000; **47**(9):31–41.
39. Abdel-Hamid TK. The dynamics of software project staffing: A system dynamics based simulation approach. *IEEE Transactions on Software Engineering* 1989; **15**(2):109–119.

DOI: 10.1002/spe

40. Brooks F. *The Mythical Man-Month 20th Anniversary Edition*. Addison-Wesley: Reading, MA, 1995.
41. Di Penta M, Harman M, Antoniol G, Qureshi F. The effect of communication overhead on software maintenance project staffing: A search-based approach. *Proceedings of the 23rd IEEE International Conference on Software Maintenance*, Paris, France, 2–5 October. IEEE Computer Society Press, 2007; 315–324.
42. Gross D, Harris C. *Foundamentals of Queueing Theory*. Wiley: New York, NY, 1998.
43. Coello Coello CA. Theoretical and numerical constraint-handling tehniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering* 2002; **191**(11–12):1245–1287.
44. Michalewicz Z, Fogel DB. *How to Solve it*: *Modern Heuristics* (2nd edn). Springer: Berlin, Germany, 2004.
45. French S. *Sequencing and Scheduling*: *An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood: New York, 1982.
46. Goldberg DE. *Genetic Algorithms in Search*, *Optimization and Machine Learning*. Addison-Wesley: Reading, MA, 1989.
47. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of state calculations by fast computing machines. *Journal of Chemical Physics* 1953; **21**:1087–1092.
48. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 2002; **6**(2):182–197.
49. De Jong KA. An analysis of the behavior of a class of genetic adaptive systems. *PhD Dissertation*, University of Michigan, Ann Arbor, MI, 1975.
50. Antoniol G, Di Penta M, Harman M. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. *10th International Software Metrics Symposium* (*METRICS 2004*), Los Alamitos, CA, U.S.A. IEEE Computer Society Press: Silver Spring, MD, 2004; 172–183.

DOI: 10.1002/spe