Adaptive Multi-Objective Evolutionary Algorithms for Overtime Planning in Software Projects

Federica Sarro, Filomena Ferrucci, Mark Harman, Alessandra Manna, and Jian Ren

Abstract—Software engineering and development is well-known to suffer from unplanned overtime, which causes stress and illness in engineers and can lead to poor quality software with higher defects. Recently, we introduced a multi-objective decision support approach to help balance project risks and duration against overtime, so that software engineers can better plan overtime. This approach was empirically evaluated on six real world software projects and compared against state-of-the-art evolutionary approaches and currently used overtime strategies. The results showed that our proposal comfortably outperformed all the benchmarks considered. This paper extends our previous work by investigating adaptive multi-objective approaches to meta-heuristic operator selection, thereby extending and (as the results show) improving algorithmic performance. We also extended our empirical study to include two new real world software projects, thereby enhancing the scientific evidence for the technical performance claims made in the paper. Our new results, over all eight projects studied, showed that our adaptive algorithm outperforms the considered state of the art multi-objective approaches in 93 percent of the experiments (with large effect size). The results also confirm that our approach significantly outperforms current overtime planning practices in 100 percent of the experiments (with large effect size).

Index Terms—Software engineering, management, planning, search-based software engineering, project scheduling, overtime, hyperheuristic, multi-objective evolutionary algorithms, NSGAII

1 Introduction

Few software engineers can have failed to notice the harmful effects of unplanned overtime on the software industry. Software engineering is notorious for effort estimate inaccuracy and time-to-market pressure, with software engineers often finding themselves coerced into high levels of unplanned overtime. It is widely believed that this leads to dissatisfaction and depression, which are worrying enough in themselves [1]. Moreover, asking people to work beyond their working hours not only increases project costs but also leads to burnout, errors, and rework [2], [3], [4], none of which is a characteristic of a successful project. In its most extreme form, unplanned overtime results in a so-called 'death march project' [5], with all the implications this inherently has for quality of software, and the quality of life of engineers unfortunate enough to find themselves involved in such projects.

Problems associated with unplanned overtime have been widely reported upon in the occupational health literature. This literature contains several systematic studies of the effects of unplanned overtime on professionals. Demanding unplanned overtime from people at a short notice could take time from their lives, disrupting their work-life balance with consequent negative effect on their morale [4]. Even from a 'purely product-focussed point of view' (divorced from any concerns over engineers' welfare), this literature also highlights the harmful impact of unplanned overtime the products and services professionals are able to provide [6], [7], [8].

Although there is a great deal more literature on the general problems of unplanned overtime in the wider workplace than there has been specific evidence focussing on software engineering projects, there is also evidence specifically concerned with software engineering professionals: A controlled study of 377 software engineers found positive correlations ($p < 0.05$) between unplanned overtime and several widely-used stress and depression indicators [2]. There is also evidence that the deployment of overtime can lead to increased software defect counts [3].

Fortunately, there is also case study evidence that proper overtime planning (i.e., allocating fairly in advance some extra amount of time to complete a certain task) leads not only to greater software engineer job satisfaction, but also to improved customer satisfaction in the resulting software products [1], [4], [9]. Indeed, thanks to overtime planning, a project manager can analyse in advance if there is any potential benefit (e.g., reduce risk of overrun) from working overtime on certain tasks rather than others and evaluate whether the current team can handle the project or some overtime is needed to cover the gap [4]. Looking to the wider (non-software-engineering specific) literature, we can also find evidence that, if overtime is *properly planned* then there are few, if any, of the harmful side-effects that so-often accompany unplanned overtime [10]. This evidence all points to the need for research into decision support for

- *F. Sarro and M. Harman are with the University College London, CREST Centre, London WC1E 6BT, United Kingdom. E-mail: {f.sarro, mark.harman}@ucl.ac.uk.*
- *F. Ferrucci and A. Manna are with the University of Salerno, Fisciano, SA 84084, Italy. E-mail: fferrucci@unisa.it, a.manna9@studenti.unisa.it.*
- *J. Ren is with Beihang University, Beijing 100191, China. E-mail: jian.ren@cs.ucl.ac.uk.*

Manuscript received 11 July 2015; revised 11 Dec. 2016; accepted 22 Dec. 2016. Date of publication 10 Jan. 2017; date of current version 23 Oct. 2017. Recommended for acceptance by T. Menzies.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2650914

software engineers to help them better plan for overtime, balancing the need for overtime against project overrun risks and budgetary constraints.

Given the importance of the problem for both software engineers and the products they produce, it is surprising that this problem has not been more widely scientifically studied. The research agenda we report on in this paper seeks to address this lack of work on overtime planning for software engineering project management.

Previously [11], we introduced an approach to support software engineers in better planning for overtime by identifying in advance when is best to work beyond normal working hours. The problem is to find the right balance between the conflicting objectives of reducing project duration, amount of overtime, and risk of project overrun. Complex multi-objective decision problems with competing and conflicting constraints such as this are well suited to Search Based Software Engineering (SBSE) [12], which has proved able to provide decision support for other early-stage software engineering activities, notably requirements engineering [13], [14], [15]. However, this is the first time that an approach has been introduced to provide decision support for software engineers attempting to reconcile the complex trade-offs inherent in overtime planning. Our previous work [11] thus introduced the first search based formulation of the project overtime planning problem.

Our approach balances trade offs between project duration, overrun risk, and overtime resources for three different risk assessment models. It is applicable to software project plans, such as those constructed using the Critical Path Method, widely adopted by software engineers and implemented in many tools. This paper extends that work, with novel adaptive algorithms for overtime planning and wider evaluation on a larger number of real world data sets concerning software project management.

Our original approach was evaluated on six real world software projects, using three standard evaluation measures and three different approaches to risk assessment. The results showed that the proposed approach is significantly better than currently used overtime planning strategies with large effect size. Moreover, they revealed that using the Non-dominated Sorting Genetic Algorithm 2 (NSGAII) with a crossover operator specifically conceived for the overtime planning problem (i.e., NSGAII$_v$) leads to significantly improvement over a standard multi-objective search (i.e., NSGAII).

In this paper we extend our previous work [11], as follows:

1) We investigate adaptive multi-objective approaches to meta-heuristic operator selection, thereby extending and (as the results show) improving the algorithmic performance of the approach proposed in the conference version of this paper (which considered only non-adaptive approaches) [11]. This is the first use of adaptive multi-objective evolutionary algorithms for software project management.
2) We validate our proposed multi-objective approach for overtime planning by using two new real projects in addition to the six ones previously used in our conference paper [11]. This leads to 288 different experiments, comparing Adaptive$_{vsc}$ the proposed algorithm with adaptive crossover selection and domain specific crossover operator to random search (a sanity check), and to the two standard multi-objective algorithms for overtime planning from the conference version of our paper. Additionally, we have compared our adaptive algorithm to the adaptive NSGAII originally proposed by Nebro et al. [16]. The results reveal that our approach is statistically significantly better than random search in 100 percent experiments (with large effect size) and is also statistically significantly better than the considered state of the art multi-objective approaches in 93 percent of experiments (with large effect size).
3) We also compare the performance of seven adaptive NSGAII variants introduced in this paper to assess the impact of using different crossover and adaptive strategies. This leads to 432 experiments: Adaptive$_{vsc}$ outperforms the other approaches in 281 cases, provided similar results in 91 cases, and was worse in only 60 cases. The results suggest that the criteria used to adaptively select the genetic operator during the search are important to obtain an effective algorithm.
4) We compare the new adaptive algorithm to standard overtime planning strategies reported in the literature. This reveals that our approach significantly outperforms these standard strategies with high effect size in all the experiments, thus confirming and extending previous results [11].

The rest of the paper is organised as follows: In Section 2 the overtime planning problem is defined. Section 3 introduces the search based approach proposed to address this problem using a multi-objective Pareto optimal approach. Section 4 describes the method used in our empirical studies, the results of which are presented in Section 5. Section 6 analyses the limitations of the present study, while Section 7 describes the context of related work in which the current paper is located. Section 8 concludes and presents directions for future work.

2 Overtime Problem Formulation

The formulation of the overtime problem we previously introduced [11] starts from the Work Breakdown Schedule (WBS) produced by the software engineer. The WBS is a hierarchical decomposition of the project goals into smaller and manageable tasks (i.e., work packages) that are executed and delivered by the project team to accomplish the project goals. The upper levels in the hierarchy represent the major project deliverables, while the lower levels depict the granular level activities needed to be performed towards achieving the deliverables. This provides project managers with a better control of all project planning activities. The WBS allows indeed to: (i) define the project scope in terms of deliverable and components; (ii) provide the framework on which the project status and progress reports are based; (iii) provide inputs for other project management processes like estimation, scheduling, staff assignment, risk assessment, etc. The number and complexity of the WBS levels depend on the size and nature of the project. Many tools such as Microsoft project [17] (the tool used by all the organisations that provided the real world schedules used

to evaluate our approach in this paper), can support software engineers to produce a WBS.

A formal definition of a WBS follows: Let a project schedule be represented as an acyclic directed graph consisting of a node set $WP = \{wp_1, wp_2, ..., wp_n\}$ of work packages and an edge set $DEP = \{(wp_i, wp_j) : i \neq j, 1 \leq i \leq n, 1 \leq j \leq n\}$ of dependencies between elements of WP, where wp_j can start only when wp_i has completed. WP and DEP form a graph, the set of paths, Π, of which, denote the dependence-respecting orderings of work packages to be undertaken to complete the project.

Associated with each work package, $wp \in WP$, is the estimated effort, e_{wp}, required to complete wp (such an effort in our study is provided for each work package by the software company and it is measured as normalized person hours). The estimated duration (in days) of a given work package $Duration(wp)$, can be computed by dividing the estimated effort e_{wp} for the number of hours worked per day on that work package.

Based on this, the duration of each path $p \in \Pi$ through the project dependence graph is given by

$$Duration_p = \sum_{\forall wp \in p} Duration(wp) \tag{1}$$

and the total estimated shortest possible duration of the project is given by any maximal length (or 'critical') path in Π. This is a formalisation of the well-known 'Critical Path Method' [18], which has been widely used in project planning for several decades. Though there may be several equal length critical paths (for which no other path is longer) it is traditional to select one and to refer to it as *the* critical path, CP [19], in our experiment if there is more than one critical path we select one of them uniformly at random.

Our problem is to analyse the effects of choice of overtime assignments, each of which seeks to minimize project duration, risk of overrun and the amount of overtime deployed. This can be formulated as a three objective decision problem in which the three objectives of duration, risk and overtime are conflicting minimisation objectives.

We represent a candidate solution to our problem as an assignment of overtime to work packages. A feasible solution is an assignment of a certain number of extra hours to each work package, denoted by $Overtime(wp_i)$ subject to the following constraint: $0 \leq Overtime(wp_i) \leq MaxOvertime(wp_i)$, where $MaxOvertime(wp_i)$ is the maximum assignable overtime to the ith work package and depends on the maximum overtime assignable per each day of its expected duration.[1]

We shall use computational search to seek an allocation of overtime for all work packages that minimises each of the three objectives of Overtime (O), Project Duration (D) and Risk of Overrun (R). We therefore measure fitness as a triple $\langle O, D, R \rangle$, whose components are defined as follows:

1. The length of a working day and maximum allowed overtime per day are country specific parameters to our approach, determined by legal and governance procedures in place. In this paper we set these to 8 hours for a working day and 3 hours maximum overtime per day. Of course different settings can be used without altering the formulation of the problem.

Overtime(O) is the amount of time worked on each work package beyond the individual time limit $Overtime(wp_i)$ summed over all work packages in WP[2]. More formally:

$$O = \sum_{i=0}^{n} Overtime(wp_i) \tag{2}$$

Project Duration (D) is the estimated duration (i.e., the length of the critical path). More formally:

$$D = \sum_{wp \in CP} Duration(wp) \tag{3}$$

We define the risk of overrun in terms of the risk of overrun associated to each path, p, in the project schedule:

$$risk_p = \frac{Duration_p}{Duration_{CP}} \tag{4}$$

The closer $risk_p$ is to 1.0, the greater the chance that an overrun on a work package along path p will cause p to supersede the current critical path as the determinant of project duration (p thus becoming the new critical path due to the overrun).

We use three different approaches to the measurement of Risk of Overrun (R), each of which combines the path risk $risk_p$, above into an overall project risk, R, as follows:

$$R = R_{AvgRisk} = \frac{\sum_{p \in \Pi} risk_p}{|\Pi|} \tag{5}$$

$$R = R_{MaxRisk} = max_{p \in \Pi - CP} risk_p \tag{6}$$

$$R = R_{TrsRisk}(L) = \frac{\left|\{p \,|\, p \in \Pi \wedge risk_p > L\}\right|}{|\Pi|} \cdot 100 \tag{7}$$

These are, respectively, average, maximal, and threshold level risks. Average risk is suited to the engineer who is 'risk averse'; it assumes that any overrun on any path could be a problem. This is 'risk averse' in the sense that it reflects a pessimistic belief that 'anything that can go wrong will go wrong'. Maximum risk is better suited to the engineer who is more concerned that the critical path is not disrupted, but who is relaxed about overruns in non-critical paths that do not threaten to supersede the critical path, as these could be absorbed into the project schedule. Threshold risk allow the engineer to choose a risk level, making risk level a parameter to the overall approach (which we set to 0.75 in this paper). These three choices seek to capture realistic instantiations of approaches that would suit a particular management style.

2. Please, note that this objective does not consider the monetary costs associated to overtime. Overtime monetary cost can be calculated by multiplying the overtime hours by the resource's overtime rate. The resource's overtime rate, however, may vary depending on the software company policies [4] (for example, some resources can be paid the same hourly rate for extra hours or a more rewarding rate, salaried employees may not be paid overtime, etc.). Where data is available the monetary cost may be used as an alternative objective function to the one defined herein. It is worth to note that even if the salaried employees are not paid to work overtime (so there is no monetary cost associated to the overtime) they are anyway demanded to do it, in this case it would be even more critical to plan overtime since for an employee is not rewarding to work beyond its regular working time for no compensation.

However, other choices are possible and we leave the investigation of these to future work.

Of course, overtime allocation is a disruptive process; it can change the critical path. This is one of the motivations for decision support: engineers cannot be expected to understand the impact of proposed overtime allocations on the critical path, while simultaneously balancing budget, duration, and estimates of overrun. These are precisely those problems for which we need the kind of automated analysis we investigate in this paper.

3 The Solution Approach

Our solution uses Search Based Software Engineering (SBSE) [12], [20], for which it is established best practice to define a representation, fitness function and computational search algorithm [21]. Since our formulation is a triple objective formulation we also need to decide how to handle multiple objectives.

3.1 Handling Multiple Objectives

It is possible to combine multiple objectives into a single criterion, however, in cases like ours where there are no obvious weights that allow a combination in a single criterion, it is recommended to use a multi-objective approach.

Since in our case, the three objectives are measured on orthogonal scales we use Pareto optimality [21], which states: A solution x_1 is said to dominate another solution x_2, if x_1 is no worse than x_2 in all objectives and x_1 is strictly better than x_2 in at least one objective.

Pareto optimality means that we do not suggest to the engineer a single proposed solution. That would not be realistic. No engineer would trust an automated tool to provide a single overtime allocation. Rather, we seek to provide a decision support tool, by showing the solutions in a space of trade offs between the three objectives, allowing the engineer to see the trade offs between them.

Using Pareto optimality we can plot the set of solutions found to be non-dominated (and therefore equally viable). In the case where there are three objectives, such as ours, this leads to a three dimensional Pareto surface, though we can also project this surface onto a two dimensional Pareto front to focus any two objectives of interest. The shapes of such surfaces and fronts can yield actionable insights. For example, where there is a knee point (a dramatic switch in the material values of trade off between objectives), this guides decision making (see Section 5).

3.2 Solution Representation

Feasible solutions to the problem defined in Section 2 are assignments of a certain number of overtime hours to each work package. We encoded them as chromosomes of length n, where each gene represents the number of extra hours assigned to each work package. The initial population, composed by n chromosomes, was randomly obtained by assigning to each wp_i an overtime ranging from 0 to $MaxOvertime(wp_i)$.

3.3 Fitness Evaluation

To evaluate the fitness of each chromosome we employed a multi-objective function to simultaneously minimise the objectives described in Section 2, namely Project Duration, Overtime, and Risk of Overrun. We report results for each overrun risk assessment measure (AvgRisk, MaxRisk, and TrsRisk) separately to explore the effects of each approach to risk assessment.

3.4 Computational Search

In our previous work [11] we employed a widely used Multi-Objective Evolutionary Algorithm, namely NSGAII [22] as ranking method to solve the multiobjective overtime planning problem. We also proposed a variant, NSGAII$_v$, which performed significantly better than NSGAII and some currently used software engineering practices tipically applied to overtime planning problems [11]. The main difference with respect to the original NSGAII lies in the use of a new crossover specifically conceived for the overtime planning problem. Indeed, it is often insufficient merely to apply a generic algorithm like NSGAII 'out of the box'; we need to define problem-specific genetic operators to ensure best performance. In the case of genetic algorithms, such as NSGAII, the crossover operator plays a pivotal role [23], [24], [25] and thus forms a natural focus for such problem-specific algorithm design.

In this paper, we investigate adaptive multi-objective approaches to meta-heuristic operator selection. This follows a recently proposed hyperheuristic SBSE approach in which the search algorithm learns the best genetic operators to be used among a given set of available operators during the search process (i.e., the search algorithms is able to adapt itself during the search) [26].

In particular, in addition to the above approaches, we investigated the use of a new adaptive version of NSGAII, namely Adaptive$_{vsc}$, which sinergically combines the crossover proposed in our previous work [11] with an extension of NSGAII, namely NSGAII$_a$, proposed by Nebro et al. [16]. The general idea is that NSGAII$_a$ works as NSGAII, but the genetic operators are selected adaptively during the search from a set of different operators [16]. With respect to the algorithm proposed by Nebro et al. [16] we extended the operator set by including a crossover we specifically conceived for overtime planning [11] and we introduced a new adaptive strategy to select the genetic operators during the search.

We analysed the effect of these choices by applying incremental changes to NSGAII$_a$, each of them resulted in a different algorithm that has been evaluated and compared against NSGAII$_a$ and its variations as detailed in the following. The final algorithm, namely Adaptive$_{vsc}$, has resulted to be superior with respect to NSGAII, NSGAII$_v$, and NSGAII$_a$ as detailed in Section 5.

The first algorithm we analysed, referred to as Adaptive$_v$ hereinafter, works as NSGAII$_a$, but includes in the operators' set a crossover we previously proposed for overtime planning [11]. We also introduced two further adaptations since we noticed that for the problem under investigation both NSGAII$_a$ and Adaptive$_v$ often converge very early (i.e., after 5 to 15 generations) on the selection of the strongest operator in the set thus preventing the use of other operators that may still be useful later on in a different phase of the search. The first adaption (denoted with the subscripted suffix 's') sorts the operators' set in ascending order each time that an early convergence is observed giving again to

TABLE 1
The Fixed and Adaptive Multi-Objectives Evolutionary Algorithms to Overtime Planning Investigated in This Work

Algorithm	Brief Description
NSGAII	The NSGAII algorithm originally proposed by Deb et al. [27].
NSGAII_v	Modified version of NSGAII that uses a crossover specifically conceived for overtime planning by Ferrucci et al. [11].
NSGAII_a	Adaptive NSGAII orginally proposed by Nebro et al. [16] that selects the genetic operator to use during the search from a predefined operator pool.
Adaptive_v	Same as NSGAII_a but includes in the genetic operator pool a crossover specifically conceived for overtime planning by Ferrucci et al. [11]
Adaptive_s	Modified version of NSGAII_a that uses a different strategy to sort the operator pool.
Adaptive_c	Modified version of NSGAII_a that uses a different adaptive criterion to select the genetic operators applied during the search.
Adaptive_{sc}	Modified version of NSGAII_a that uses the sorting criterion and the adaptive criterion used by Adaptive_s and Adaptive_c, respectively.
Adaptive_{vs}	Same as Adaptive_s but includes in the genetic operator pool a crossover specifically conceived for overtime planning by Ferrucci et al. [11]
Adaptive_{vc}	Same as Adaptive_c but includes in the genetic operator pool a crossover specifically conceived for overtime planning by Ferrucci et al. [11]
Adaptive_{vsc}	Same as Adaptive_{sc} but includes in the genetic operator pool a crossover specifically conceived for overtime planning by Ferrucci et al. [11]

all the operators equal probability to be selected (this probability is subsequently updated during the search in the same way as for NSGAII_a). The second one (denoted with the subscripted suffix 'c') uses instead a different adaptive selection criterion. We analysed the effect of applying these adaptions alone or combined, thus considering the following variants: Adaptive_s, Adaptive_c, Adaptive_{sc}, Adaptive_{vs}, Adaptive_{vc}, Adaptive_{vsc}.

In the following we provide details about each of the algorithms we investigate in this paper. For the sake of clarity, we first present the original techniques (i.e., NSGAII, NSGAII_v, NSGAII_a) and then the new variants we introduce in this paper. Table 1 summarise the key aspects of each algorithm. We use the "stem" name NSGAII for the 3 existing algorithms and "Adaptive" for the novel variations in order to clarify which is novel to this paper from which is studied for comparison.

- *NSGAII* decomposes the population into several fronts, as follows: ① all the solutions are ranked using the non-dominance concept; ② all non-dominated solutions of the population are assigned to rank 1, then they are removed from the population; ③ iteratively, non-dominated solutions are determined and assigned rank 2. Steps 1-3 are iterated until the population is empty. Then the solutions are ranked again according to a crowding distance, namely the difference between the left and right neighbors or infinity if there are no neighbors. The use of the crowding distance is crucial to preserve the diversity in the solutions fronts, since computing the distance between a given solution and its nearest neighbors allows NSGAII to approximate the density of the obtained solution. So, solutions with higher crowding distance are considered better solutions, as they introduce more diversity in the population. Once all the solutions are ranked by both dominance and crowing distance, crossover and mutation operators are applied to produce an offspring. Then a tournament selector is applied and the best m solutions (in terms of dominance and crowding) are copied into the next generation. The algorithm is stopped after a fixed number of fitness evaluations (see Section 4.5). The NSGAII used herein is the same we used in our previous work [11] and has been implemented by using the JMetal framework [28].
- *NSGAII_v* is a variant of NSGAII introduced specifically for the overtime planning problem [11]. NSGAII_v exhibits the same selection and crowding distance characteristics as the standard NSGAII but exploits a new crossover operator. This operator aims to preserve genes shared by the fittest overtime assignments, thereby avoiding the well-known disruptive effects of crossover [23]. It is defined as follows: Let P_1 and P_2 be parent chromosomes, C the point of cut randomly selected in the parents, and O_1 and O_2 the new offspring. For the genes placed before C, O_1 and O_2 inherit the genes of P_1 and P_2, respectively. While each gene g_i placed after C inherit genes from P_1 and P_2 as follows[3]:

$$O_1(g_i) = \begin{Bmatrix} P_1(g_i), p{=}0.5 \\ P_2(g_i), p{=}0.5 \end{Bmatrix}$$
$$O_2(g_i) = (P_1(g_i) + P_2(g_i))/2,$$

Note that when the parent genes hold the same characteristic (i.e., same quantity of overtime) they are retained in both offspring, otherwise we generate two different genes for the offspring: one that inherits the gene from mother or father with equal probability and one that inherits both parent characteristics in terms of overtime average. It is important to note that in multi-objective optimization, it is better to create children that are close to their parents in order to have a more efficient search process [27]. The NSGAII_v used herein is the same we used in our previous work [11] and has been implemented by using JMetal [28].

3. This definition is a simplified but equivalent version of the one given in our previous work [11]

- *NSGAII$_a$* works as NSGAII, but the genetic operators are selected adaptively during the search from a set of different operators [16] depending on the success of each operator in the past. The success is quantified as the number of children produced by each operator that survived to the next generation. The employed operator set is composed of three genetic operators commonly used in multi-objective optimization metaheuristics: the SBX crossover, the polynomial-based mutation, and the variation operator used for Differential Evolution [16]. The adaption is obtained by giving to the operators a higher probability of being chosen when they are capable of producing solutions that survive from one generation to the next. The adaptive scheme used for the operator selection is the same used in previous work [16], [29] and it is shown in Algorithm 2, while Algorithm 1 shows the pseudocode of NSGAII$_a$ [16]. NSGAII$_a$ proceeds as follows. For each individual in the population, it produces a random value in [0,1] (Algorithm 1 line 6) and computes the contribution (i.e., `probabilityOperator`) of each genetic operator (line 7) using the method shown in Algorithm 2. These values are compared and one out of the three genetic operators is selected (Algorithm 1 lines 7-18) as follows: If the random value is less than the probability value associated to the first genetic operator, this operator is selected to create the new offspring, otherwise the second genetic operator is compared, and so on. If no condition is satisfied, the last operator is selected. The `probabilityOperator` values are computed by using the adaptive selection scheme shown in Algorithm 2. This method takes as input the set of different genetic operators and a population P in order to compute the contribution of each of the genetic operators, i.e., how many solutions generated by each operator are part of the next generation of the population (Algorithm 2: line 3). If an operator has contributed with less solutions than a minimum threshold, its contribution is set to this minimum threshold (Algorithm 2: lines 4-6). This helps us prevent discarding an operator if it does not produce any survival solutions in a certain iteration. In our work we have considered a threshold equal to 2 as suggested in previous work [16], [29]. Once the contribution of all operators has been computed, the `probabiltyOperator` is returned to the main procedure (Algorithm 2: line 12). Then, the NSGAII$_a$ algorithm behaves as the original NSGAII (Algorithm 1 lines 19-26). A reference implementation of NSGAII$_a$ is freely available in JMetal [28] since Version 4.5.
- *Adaptive$_v$* works as NSGAII$_a$ but we added the crossover operator used in [11] to the genetic operator set already used by NSGAII$_a$ [16]. To realize Adaptive$_v$ (and also the adaptions described in the following) we extended the implementation of NSGAII$_a$ provided in JMetal [28].
- *Adaptive$_s$* (*Adaptive$_{vs}$*): works as NSGAII$_a$ but every time an early convergence is detected the genetic operators in the set are sorted in ascending order with respect to the their success (i.e., the number of children that survived to a next generation) and equal probability is given to each of them before continuing the search by applying the same adaptive selection criterion as used in NSGAII$_a$ (Adaptive$_v$, respectively).

  Adaptive$_c$ (*Adaptive$_{vc}$*) works as NSGAII (Adaptive$_v$, respectively) but adaptively chooses the genetic operators at each iteration depending on the quality of the offspring generated by each operator in terms of each of the single objectives included in the multi-objective formulation (Section 2). It works as follows: At each iteration, all the children that survive to the next generation are compared against the best individuals survived to the previous generations and every time a child is better than at least an existing individual for a given objective, the success of the genetic operator that produced that child is incremented. Once the success of each operator has been updated, for each individual in the population, a random value in [0,1] is produced and compared with the operators success in the last iteration. Depending on this value, one of the genetic operators is selected. In case the random number is higher than the success of all the operators, the genetic operator with the highest success is chosen. Once the offspring is generated, the algorithm behaves as the original NSGAII.
- *Adaptive$_{sc}$* (*Adaptive$_{vsc}$*) sorts the genetic operators in the same way as Adaptive$_s$ before applying the adaptive selection criterion used for Adaptive$_c$ (Adaptive$_{vc}$).

Algorithm 1. Pseudocode of NSGAII Adaptive (NSGAII$_a$) [16]

Require: n, operatorList {n = population size, operatorList = set of genetic operators}
1: $P \leftarrow RandomPopulation()$ {P = population}
2: $Q \leftarrow \emptyset$ {Q = auxiliary population}
3: **while** $notTerminationCondition()do$ **do**
4: **for** $i \leftarrow 1$ to n **do**
5: $randValue \leftarrow rand()$
6: $probabilityOperator[] \leftarrow contribution(P, operatorList)$ {See Algorithm 2 for the definition of the contribution() method}
7: **if** $randValue \leq probabilityOperator[0]$ **then**
8: $parents \leftarrow Selection2(P)$
9: $offspring \leftarrow SBX(parents)$
10: **else**
11: **if** $randValue \leq probabilityOperator[1]$ **then**
12: $parents \leftarrow Selection3(P)$
13: $offspring \leftarrow DE(parents)$
14: **else**
15: $parents \leftarrow Selection1(P)$
16: $offspring \leftarrow PolynomialMutation(parents)$
17: **end if**
18: **end if**
19: $EvaluateFitness(offspring)$
20: $Insert(offspring, Q)$
21: **end for**
22: $R \leftarrow P \cup Q$
23: $RankingAndCrowding(R)$
24: $P \leftarrow SelectBestIndividuals(R)$
25: **end while**
26: **return** P

Algorithm 2. Computing the Contribution of Each Genetic Operator [16], [29]

Require: P, operatorList {P = population, operatorList = set of genetic operators}
1: $totalContribution \leftarrow 0$
2: $numOperators \leftarrow size(operatorList)$ numOperators indicates the number of operators in the pool
3: **for** $operator \leftarrow 0$ to $numOperators$ **do**
4: $contributionOperator[operator] \leftarrow solutionInNextPopulation(P, operatorList[operator])$
5: **if** $contributionOperator[operator] \leq threshold$ **then**
6: $contributionOperator[operator] \leftarrow threshold$
7: **end if**
8: $totalContribution \leftarrow totalContribution + contributionOperator[operator]$
9: **end for**
10: **for** $operator \leftarrow 0$ to $numOperators$ **do**
11: $probabilityOperator[operator] \leftarrow contributionOperator[operator]/totalContribution$
12: **end for**
13: **return** $probability\ Operator$

4 The Design of the Empirical Study

This section explains the design of our empirical study; the research questions we set out to answer and the methods and statistical tests we used to answer these questions. We adopted the same methodology, the same risk assessment measures and the same quality indicators we used in the original study [11]. However we extended the study by adding two new datasets to the six ones previously used [11].

4.1 Research Questions

We seek to answer the following four research questions. Three of them (i.e., RQ1, RQ3, and RQ4) were posed in our previous work [11] and are answered herein to assess the effectiveness of the adaptive multi-objective algorithms we introduced in this work.[4] We also answered a new question (i.e., RQ2) to investigate the different variants of NSGAII$_a$.

RQ1 (SBSE Validation). How do NSGAII$_a$ and its variants perform compared to random search? In any attempt at an SBSE formulation of a problem this is a standard baseline question asked. If a proposed formulation does not allow an intelligent computational search technique to outperform random search convincingly, then there is clearly something wrong with the formulation. This question is thus adopted in SBSE research as a preliminary 'sanity check' [30]. We therefore compare the adaptive evolutionary algorithms proposed in this work with respect to random search.

RQ2 (Comparison of Different NSGAII$_a$ Adaptions). How does the use of the adaptions we introduced affect the performance of NSGAII? Since in our previous work [11] we showed that using a crossover operator specifically designed for the overtime problem leads to significant improvement of NSGAII's performance, in this paper we studied adaptive multi-objective approaches to meta-heuristic operator selection. In particular, we analysed the adoption of this crossover in combination with a new algorithm, namely NSGAII$_a$, that is able to adaptively select the genetic operators during the search from a pool of different operators. Therefore, to answer this goal we compare on the considered datasets the performances of all the NSGAII$_a$ variants introduced in Section 3.4 (i.e., Adaptive$_s$, Adaptive$_c$, Adaptive$_{sc}$, Adaptive$_{vs}$, Adaptive$_{vc}$, Adaptive$_{vsc}$) .

RQ3 (Comparison to State of the Art Search). How does Adaptive$_{vsc}$ perform compared to the state of the art? Outperforming random search is necessary, but not sufficient. In order for a proposed approach to be adopted it must also outperform the state of the art for the problem in hand. In our case the state of the art is represented by NSGAII and NSGAII$_v$ used in our previous work [11] for the overtime problem and NSGAII$_a$ [16] that we first use herein to the overtime planning problem.

RQ4 (Usefulness). How does Adaptive$_{vsc}$ perform compared to currently used overtime planning approaches? While outperforming a standard multi-objective search may be a valuable technical result, in order to be useful to software engineers, our approach must also outperform existing overtime management strategies used by practicing software engineers. We therefore repeat the experiments in RQ1 and RQ3, but for RQ4 we compare our approach with three currently used strategies.

4.2 Software Projects Used in the Empirical Study

To assess the performance of our approach we used 8 datasets representing real world software projects. All of them have been used in previous work on project staffing and scheduling (see, e.g., [31], [32], [33]) and six of them have been also used in previous work on multi-objective software project overtime planning[5] [11].

The projects came from eight different organisations, involved different kinds of software engineering development, and had different sizes, ranging from 31 to 245 work packages and from a few person weeks to several person years in duration.

Table 2 summarises the key information concerning the 8 datasets here described in details:

`DB2` concerned the next release of a large data-intensive, multi-platform software system, written in several languages including DB II, SQL, and .NET.

`Web` delivered a web-based IT sales system across North America. The project included the development and testing of website, search engine and order management and tracking.

`Quote` was a system developed for a large Canadian sales company to provide on-demand conversion of quotes to orders. This change was both internal and customer facing and ultimately affected every level of the organisation (Web, internal development, database, sales and customers).

4. In our previous work [11] we also investigated how the Pareto-fronts obtained using multi-objective overtime planning reveal insights into the trade off between risk, duration and overtime. The same results hold for the present work.

5. Five of these datasets (those for which non-disclosure agreement allows us to publish data after publication) will be made publicly available once the paper has been published. We make this data available at http://www0.cs.ucl.ac.uk/staff/F.Sarro/projects/overtime/ to support replication and comparison of future work with our results.

TABLE 2
Software Projects Used in the Empirical Study

Project	#WPs	#Dep.	Effort	Brief Description
DB2	120	102	594	A multi-platform database upgrade involving several languages such as DB2, SQL and .NET
Web	245	247	6,664	A web-based purchase order system development
Quote	60	64	547	An enhancement of an existing system to include on-demand conversion of quotes to orders
Oracle	106	105	5,390	A large-scale Oracle database migration with tight data security constraints
Price	72	71	1,570	A client-side sales system upgrade to offer additional features to users
CutOver	95	68	2,356	Details cannot be revealed because of a Non Disclosure Agreement with the project data provider
Broker	31	40	2,192	Software project to develop a management system for a broker company
Chartwell	41	29	6,680	Software for an online gaming and gambling industry developed using several languages (Java, Flash, AS)

Effort is measured in normalised person hours.

Oracle was large scale database upgrade, migrating an old, yet mission-critical, Oracle system. The information that was migrated had an estimated value to the organisation of several million dollars and formed the cornerstone of the organisation's operations. About half of the project involved taking precautions against possible causes of data loss. This project primarily involved the Database Administration section of the organisation. However, the Software Application Development section was also involved at the end for training and for upgrading the existing scripts and triggers to make use of the newly available database functionality.

Price was an enhancement to the client side of a sales system to provide improved pricing and features for discounts, vouchers, and price conversions. The enhancement concerned has a potentially significant influence on the organisation's revenue stream, so extensive QA was involved. This project involved the web portion of the company's infrastructure with smaller impact on the underlying database and other internal software. The project concluded with an employee training phase.

The details of project CutOver are the subject of a Non-Disclosure agreement and so cannot be published.

Broker was a software project developed by an IT company to implement a vessel insurance policies management system for a large insurance broker company. The project consisted of 4 management tasks, 2 database design tasks, 21 programming tasks, and 4 testing tasks [33].

Chartwell was a software for an on-line gaming and gambling industry developed using several technologies, such as Java, Flash, and Action Script, by a global software team [32].

4.3 Multi-Objective Evaluation Measurements Used

Assessing the performance of a computational search algorithm for a single objective optimisation problem typically requires observations about the best solution found. This approach is not applicable for multi-objective optimisation problems because there are a set of candidate solutions, each of which is said to be 'non-dominating'. That is, each is incomparable to the others because no other solution has better values for all objectives.

Analysis of graphical plots of the solutions can provide some indications of performance, but it provides a qualitative evaluation and cannot provide a quantitative assessment of the quality of solutions from one approach relative to another. A robust evaluation requires that qualitative evaluations be augmented by a more quantitative evaluation.

To provide this quantitative assessment we employ three solution set quality indicators, namely Contributions (I_C), Hypervolume (I_{HV}), and Generational Distance (I_{GD}), as done in previous work [11]. To compute these we normalise fitness values to avoid unwanted scaling effects [28] and compute a reference front of solutions, RS, which is the set of non-dominated solutions found by the union of all approaches compared [34].

The I_C quality indicator is the simplest measure. It measures the proportion of solutions given by an algorithm, A, that lie on the reference front RS [35]. The higher this proportion, the more A contributes to the best solutions found by the approaches compared, and so the better is the quality of its solutions. I_C is a simple and intuitive measure, but it is affected by the number of solutions produced, unfavourably penalising algorithms that might produce 'few but excellent' solutions. This is why we also consider two other measures of solution quality, I_{HV} and I_{GD}.

The I_{HV} quality indicator [36] calculates the volume (in the objective space) covered by members of a non-dominated set of solutions from an algorithm of interest. The larger this volume, the better the algorithm, because the more it captures of the non-dominated solution space. Zitzler demonstrates [37] that this hypervolume measure is also strictly 'Pareto compliant'. That is, the hypervolume of A is higher than B if the Pareto set of A dominates that of B. By using a volume rather than a count, this measure is also less susceptible to bias when the numbers of points on the two compared fronts are very different.

The I_{GD} quality indicator [38] computes the average distance between the set of solutions, S, from the algorithm measured and the reference set RS. The distance between S and RS in an n objective space is computed as the average n-dimensional euclidean distance between each point in S and its nearest neighbouring point in RS. We can think of I_{GD} as the distance between the front S and the reference front RS in the n-dimensional objective space of the problem.

4.4 Inferential Statistical Test Methods Used

Due to the stochastic nature of evolutionary algorithms, best practice requires the use of careful deployment of inferential statistical testing to assess the differences in the performance of the algorithms used [21], [39]. We therefore performed 30 independent runs per algorithm, per risk assessment measure, and per project to allow for such statistical testing.

To analyse the normality of distributions we employed the Shapiro-Wilks test [40]. As we expected, many of our

samples showed no evidence that they come from normally distributed populations, making the t-test unsuitable. We therefore used the Wilcoxon test [41] to check for statistical significance. Using the Wilcoxon test is a safe test to apply (even for normally distributed data), since it raises the bar for significance, by making no assumptions about underlying data distributions. We set the confidence limit, α, at 0.05 and applied the standard Bonferroni correction (α/K, where K is the number of hypothesis) in cases where multiple hypothesis were tested.

As has been previously noted in advice on statistical testing of algorithms such as these [21], [39], it is inadequate to merely show statistical significance alone; we also need to know whether the effect size is worthy of interest.

To this end we used the Vargha-Delaney effect size $\hat{A}_{12}$ [42], the results of which are values between 0 and 1: when the $\hat{A}_{12}$ measure is exactly 0.5, then the two compared techniques achieve equal performance; when $\hat{A}_{12}$ is less than 0.5, the first technique is worse; and when $\hat{A}_{12}$ is more than 0.5, the second technique is worse. The closer to 0.5, the smaller the difference between the techniques; the farther from 0.5, the larger the difference [42]. Given the first algorithm performing better than the second, $\hat{A}_{12}$ is considered small for $0.6 \leq \hat{A}_{12} < 0.7$, medium for $0.7 < \hat{A}_{12} < 0.8$, and large for $\hat{A}_{12} \geq 0.8$, although these thresholds are somewhat arbitrary.

Since our measurement concerns quality measures such as hypervolume and distance to the reference front, there is no domain specific transformation required for the effect size measurement [43]. Indeed, no domain specific transformation is required when (as in our case) one is interested in the fact that the probability of one technique outperforms the other.

To answer RQ1 we implemented a random search to be compared with the evolutionary approaches considered in this study (see Table 1). The random search assigns randomly to each work package of the project, an overtime varying from 0 to the maximum overtime assignable. The resulting Pareto fronts were compared for statistically significant differences with those produced by the evolutionary algorithms, using the quality indicators explained in Section 4.3. In this sanity check we used the Wilcoxon test and for significance also performed a Vargha-Delaney effect size test in all results.

We do not wish to devote too much space to RQ1, since it is only a 'sanity check', preferring to devote more space to the answers to RQs 2–4, which concern more scientifically important evidence for the performance and usefulness of our approach.

To answer RQ2 we compared the performances of all the considered NSGAII$_a$ variants (see Section 3.4) in terms of the quality indicators for statistical significance and effect size, as for RQ1, but additionally presenting the results using boxplots to give a pictorial account of the distributions of results obtained. To answer RQ3 we compared the best approach identified by RQ2 (i.e., Adaptive$_{vsc}$) with respect to the current multi-objective state of the art (i.e., NSGAII, NSGAII$_v$, and NSGAII$_a$). To answer RQ4 we repeated the same experiments and analysis performed for RQ3, but we compared Adaptive$_{vsc}$ to standard overtime management strategies. That is, we implemented three strategies currently used, and compared the results to Adaptive$_{vsc}$ using the same tests as we performed to answer RQ1 and RQ3.

TABLE 3
Configurations Explored to Tune the Nine Algorithms

Configuration	Pop. Size	Generations	Fitness Evals
Very Small (VS)	50	5,000	250,000
Small (S)	100	2,500	250,000
Medium (M)	200	1,250	250,000
Large (L)	500	500	250,000
Very Large (VL)	1,000	250	250,000

4.5 Parameter Tuning and Setting

An often overlooked aspect of research on computational search algorithms lies in the selection and tuning of the algorithmic parameters, which is necessary in order to ensure fair comparison, but which often goes unreported and, thereby, hinders any potential replication. In order to facilitate replication of our findings, in this section we report the method adopted for algorithmic parameter tuning and selection, which is a replication of the methodology previously adopted [11].

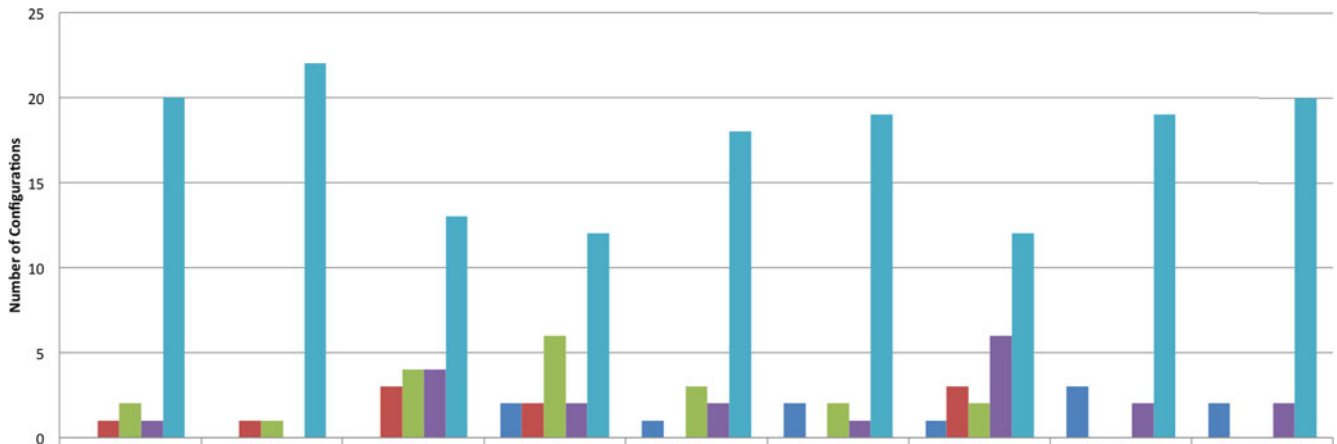
For each algorithm we evaluated five different configurations, characterised by very small (VS), small (S), medium (M), large (L), and very large (VL) values for population as detailed in Table 3. All configurations were allowed an identical budget of fitness evaluations (250,000), thereby ensuring that all require the same computational effort, though they may differ in parameter settings. We executed all the considered algorithms (see Section 3.4) with each configuration 30 times and collected the corresponding I_C, I_{HV}, and I_{GD} values, testing for significant differences using the Wilcoxon Test. Fig. 1 shows the best configurations obtained per each algorithm, over all the risk measures and datasets considered in our study. We can observe that in general a very large (VL) configuration is used in the majority of the cases, however the adaptive evolutionary algorithms require a VL configuration less often than traditional NSGAII algorithms. We run each of the algorithm with these configurations in answer our RQs.[6]

The rest of our parameter settings for both algorithms were typical standard settings. We report them here for completeness and replicability.

For population size n, at each generation, $n/2$ applications of the single point crossover operator are used by NSGAII and NSGAII$_v$ to construct offspring. As for NSGAII$_a$ and its variants the operator is adaptively chosen during the search from a set of different crossovers as explained in Section 3.4. The mutation operator randomly assigns a new value between 0 and $MaxOvertime(wp)$. The crossover and mutation operators are applied with a probability of 0.5 and 0.1, respectively.[7]

6. In order to allow for replication we report in Appendix the configurations obtained for each algorithm per risk measure and per dataset.

7. The crossover and mutation rates used in our experiment fall in the ranges recommended in previous work on search-based project management (i.e., from 0.45 to 0.95 for crossover rate and from 0.06 to 0.1 for mutation rate) [44], [45]. The impact of different settings may be investigated in future work.



	NSGAII	NSGAII_v	NSGAII_a	Adaptive_s	Adaptive_c	Adaptive_sc	Adaptive_vs	Adaptive_vc	Adaptive_vsc
VS	0	0	0	2	1	2	1	3	2
S	1	1	3	2	0	0	3	0	0
M	2	1	4	6	3	2	2	0	0
L	1	0	4	2	2	1	6	2	2
VL	20	22	13	12	18	19	12	19	20

Fig. 1. Best obtained configurations per algorithm (over 3 risk measures and 8 datasets).

We employed binary tournament selection based on dominance and crowding distance, and in tied tournaments one of the two competitor parents is chosen at random (with equal probability for both).

5 Analysis of Results

This section presents the results obtained from our experiments for RQs 1–4 set out in Section 4.1.

5.1 Results for RQ1 (SBSE Validation)

We observed that all the adaptive multi-objective approaches we considered achieved superior values with respect to random search on all the eight projects in terms of the considered quality indicators (i.e., I_C, I_{HV}, I_{GD}). The Wilcoxon Test (with Bonferroni correction) revealed that the indicator values achieved by NSGAII$_a$ and its adaptions were significantly better than those of the random search, with a 'large' $\hat{A}_{12}$ effect size for all the 24 comparisons (three risk measures, eight datasets). Thus, we conclude that there is strong empirical evidence that NSGAII$_a$ and its variations pass the sanity check denoted by RQ1.[8]

5.2 Results for RQ2 (Comparison of Different NSGAII$_a$ Adaptions)

To answer RQ2 we compared the performance of all the NSGAII$_a$ variants introduced in Section 3.4 (i.e., Adaptive$_s$, Adaptive$_c$, Adaptive$_{sc}$, Adaptive$_{vs}$,Adaptive$_{vc}$,Adaptive$_{vsc}$).[9]

Table 4 reports the mean values of each of the three quality assessment indicators obtained for 30 runs of all projects using the considered algorithms. We can observe that Adaptive$_{vsc}$ provided better values for all performance indicators with respect to other approaches in all but two cases (i.e., I_c values for TrsRisk for projects DB2 and Web where it obtained lower values than Adaptive$_{vc}$ and Adaptive$_{sc}$, respectively).

We applied the Wilcoxon Test to assess whether this difference was statistically significant on each of the datasets. In particular, for each pair (x, y) of variants we verified the following hypothesis: "The quality measure values provided by Adaptive$_x$ are significantly better than those provided by Adaptive$_y$" by taking into account three risk measures, three quality measures, and eight datasets. Since we performed multiple statistical tests, the Bonferroni correction has been used to ensure that we retain only a maximum 0.05 probability of Type 1 error.

To summarise the results of the Wilcoxon comparisons, we use the following win-tie-loss procedure [46] ; if the distribution i is statistically significantly better (less) than j according to the Wilcoxon test we updated win_i and $loss_j$, otherwise we incremented tie_i and tie_j.

Fig. 2 reports the percentage of win-tie-loss values achieved by the algorithms with different risk measures over all datasets for the MaxRisk, AvgRisk, and TrsRisk. This graphically illustrates the difference in relative performance of each of the different NSGAII$_a$ variants.

We can observe that Adaptive$_{vsc}$ provides us the best balance among win-tie-loss for all risk measures (i.e., 281-91-60), followed by Adaptive$_{sc}$ (i.e., 221-97-114), while the worst performance was achieved by Adaptive$_s$ (i.e., 81-150-201). Let us recall that the main difference between Adaptive$_s$ and Adaptive$_{sc}$ is the criteria adopted to select the genetic operator during the search (see Section 3), while Adaptive$_{vsc}$ works as Adaptive$_{sc}$ but adds to the set of genetic operators available during the search a crossover operator specifically conceived for the overtime problem [11]. Thus, the above results suggest that the criteria used to adaptively select the crossover during the search is important to obtain an effective overtime planning algorithm. Moreover, the use of a crossover specifically designed to the problem in hand allows us to significantly improve the algorithm performance.

8. The results of each algorithm can be found at http://www0.cs.ucl.ac.uk/staff/F.Sarro/projects/overtime/

9. Please, note that for brevity, we excluded Adaptive$_v$ from our analysis since we noticed that it shows the same behaviour as NSGAII$_a$. We observed that this was due to the fact that both algorithms converged early (i.e., after 5-15 generations) on the selection of the strongest operator in the pool implying that any additional genetic operator was not taken into account during the search.

TABLE 4
RQ2: Mean Values and Standard Deviation (Mean;Stdev) of the Quality Indicators for All the $NSGAIIa$ Variants (the Leading Zero Is Not Shown, e.g. 0.16 Is Reported as 16)

Project	Risk Measure	I_C							I_{HV}							I_{GD}						
		A_v	A_s	A_c	A_{sc}	A_{vs}	A_{vc}	A_{vsc}	A_v	A_s	A_c	A_{sc}	A_{vs}	A_{vc}	A_{vsc}	A_v	A_s	A_c	A_{sc}	A_{vs}	A_{vc}	A_{vsc}
DB2	MaxRisk	16;01	16;01	10;01	11;01	18;00	17;02	11;01	56;00	56;00	56;00	56;00	56;00	56;00	56;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
	AvgRisk	17;02	17;02	10;01	11;01	18;01	16;02	10;01	56;00	56;00	58;00	56;00	56;00	56;00	56;00	00;00	00;00	00;00	00;00	00;00	06;00	00;00
	TrsRisk	19;01	18;01	08;01	08;01	20;00	16;03	10;03	56;00	56;00	56;00	56;00	56;00	55;01	56;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
Web	MaxRisk	00;00	00;00	12;08	47;26	00;00	01;04	39;33	56;00	56;00	58;00	58;00	56;00	36;19	58;01	00;00	00;00	00;00	00;00	00;00	05;00	00;00
	AvgRisk	00;01	00;01	04;04	16;08	01;02	38;30	41;31	53;03	55;04	62;02	65;02	53;01	60;12	65;02	04;00	04;00	01;00	01;00	05;01	02;00	01;00
	TrsRisk	01;01	01;01	19;13	48;17	01;04	04;14	23;22	54;01	54;01	57;01	58;01	55;01	34;19	56;01	01;00	01;00	01;00	01;00	01;00	04;02	00;00
Quote	MaxRisk	14;02	13;00	13;03	13;03	14;02	20;06	13;03	57;00	57;00	57;00	57;00	57;00	56;00	57;00	01;00	01;00	01;00	01;00	01;00	01;00	01;00
	AvgRisk	10;06	10;06	15;03	13;02	16;06	20;06	06;04	56;01	56;01	57;00	56;00	56;00	56;01	57;00	05;00	05;00	05;00	04;00	05;00	04;00	05;00
	TrsRisk	05;02	06;03	06;03	21;05	10;04	20;12	31;04	58;00	58;00	58;00	58;00	58;00	56;02	58;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
Oracle	MaxRisk	10;06	12;06	13;05	20;07	00;00	25;14	19;07	51;01	51;01	51;00	51;00	45;01	50;01	51;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
	AvgRisk	00;00	00;00	19;04	22;05	00;01	32;09	26;07	45;02	49;02	54;02	54;02	57;02	54;04	55;02	01;00	01;00	01;00	01;00	01;00	01;00	01;00
	TrsRisk	12;03	02;01	07;03	09;03	20;04	00;01	46;03	50;00	49;00	50;00	50;00	50;00	44;05	50;00	00;00	00;00	00;00	00;00	00;00	01;00	00;00
Price	MaxRisk	02;01	02;01	21;34	25;03	02;01	27;06	21;02	58;00	58;00	00;00	58;00	57;00	58;02	58;00	00;00	00;00	58;00	00;00	00;00	00;00	00;00
	AvgRisk	02;00	02;00	22;03	23;02	04;01	22;08	25;03	57;01	57;00	58;00	59;00	58;00	57;04	59;00	02;00	02;00	00;00	00;00	01;00	01;00	00;00
	TrsRisk	03;01	03;01	04;01	05;01	04;01	18;10	61;07	58;00	58;00	58;00	58;00	58;00	55;04	58;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
CutOver	MaxRisk	07;04	07;03	19;02	19;04	10;04	11;09	27;04	54;00	54;00	54;00	54;00	54;00	40;19	54;00	00;00	00;00	00;00	00;000	00;00	07;10	00;00
	AvgRisk	00;00	00;00	09;06	15;05	01;02	54;07	20;04	48;02	48;01	53;01	54;01	49;01	54;00	54;01	12;01	20;02	04;00	04;00	09;07	05;00	05;02
	TrsRisk	05;03	04;02	22;08	24;08	05;01	19;15	21;07	52;01	52;01	53;00	53;00	52;01	40;20	53;00	01;00	01;00	01;00	01;00	01;01	09;08	01;00
Broker	MaxRisk	14;01	13;01	14;01	17;01	14;01	13;04	14;01	54;00	54;00	54;00	54;00	54;00	53;01	54;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
	AvgRisk	09;03	08;03	18;01	18;02	09;02	17;05	20;01	54;00	54;00	54;00	54;00	54;00	54;01	54;00	01;00	00;00	00;00	00;00	00;00	00;00	00;00
	TrsRisk	14;08	14;02	13;01	13;01	14;01	15;01	16;01	54;00	54;00	54;00	54;00	54;00	54;00	54;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
Chartwell	MaxRisk	18;07	18;09	23;06	04;06	14;06	02;02	21;06	52;00	52;00	53;00	52;00	52;00	52;00	53;00	00;00	00;00	01;00	01;00	01;00	01;00	00;00
	AvgRisk	02;02	02;02	25;10	18;08	04;05	04;02	45;14	51;00	51;00	53;00	53;00	51;00	53;00	58;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00
	TrsRisk	08;04	09;05	17;07	16;07	10;06	04;03	35;13	50;00	51;00	53;00	53;00	51;00	52;00	53;00	00;00	00;00	00;00	00;00	00;00	00;00	00;00

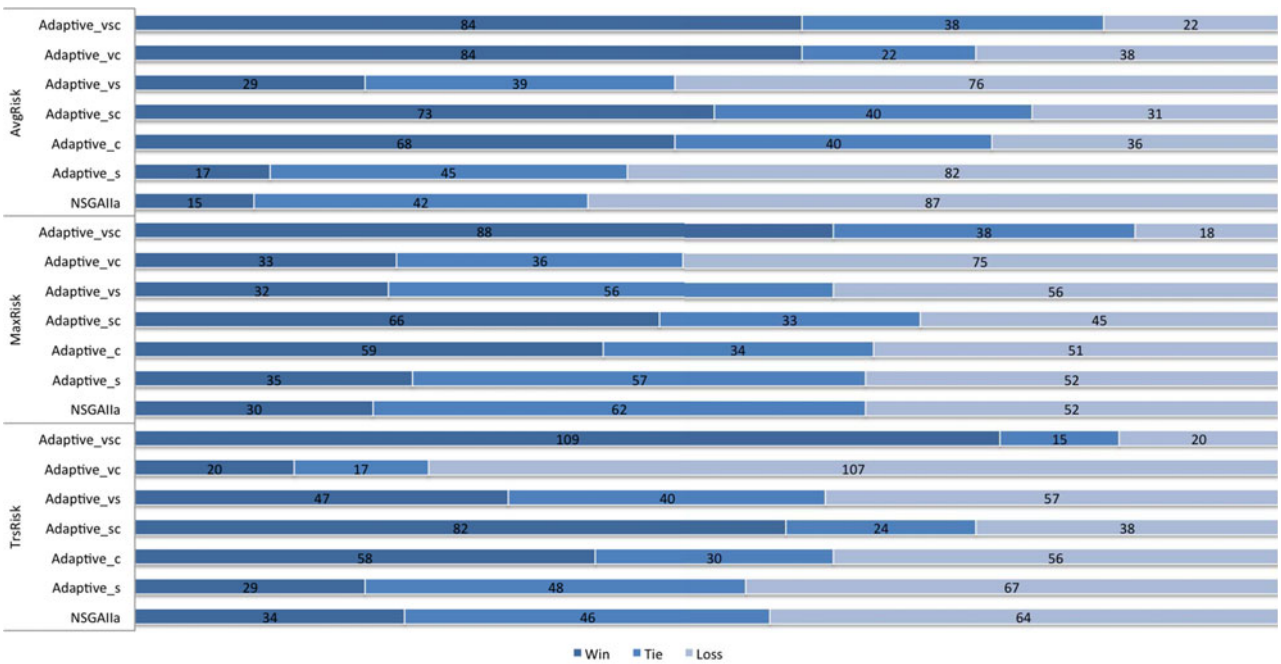
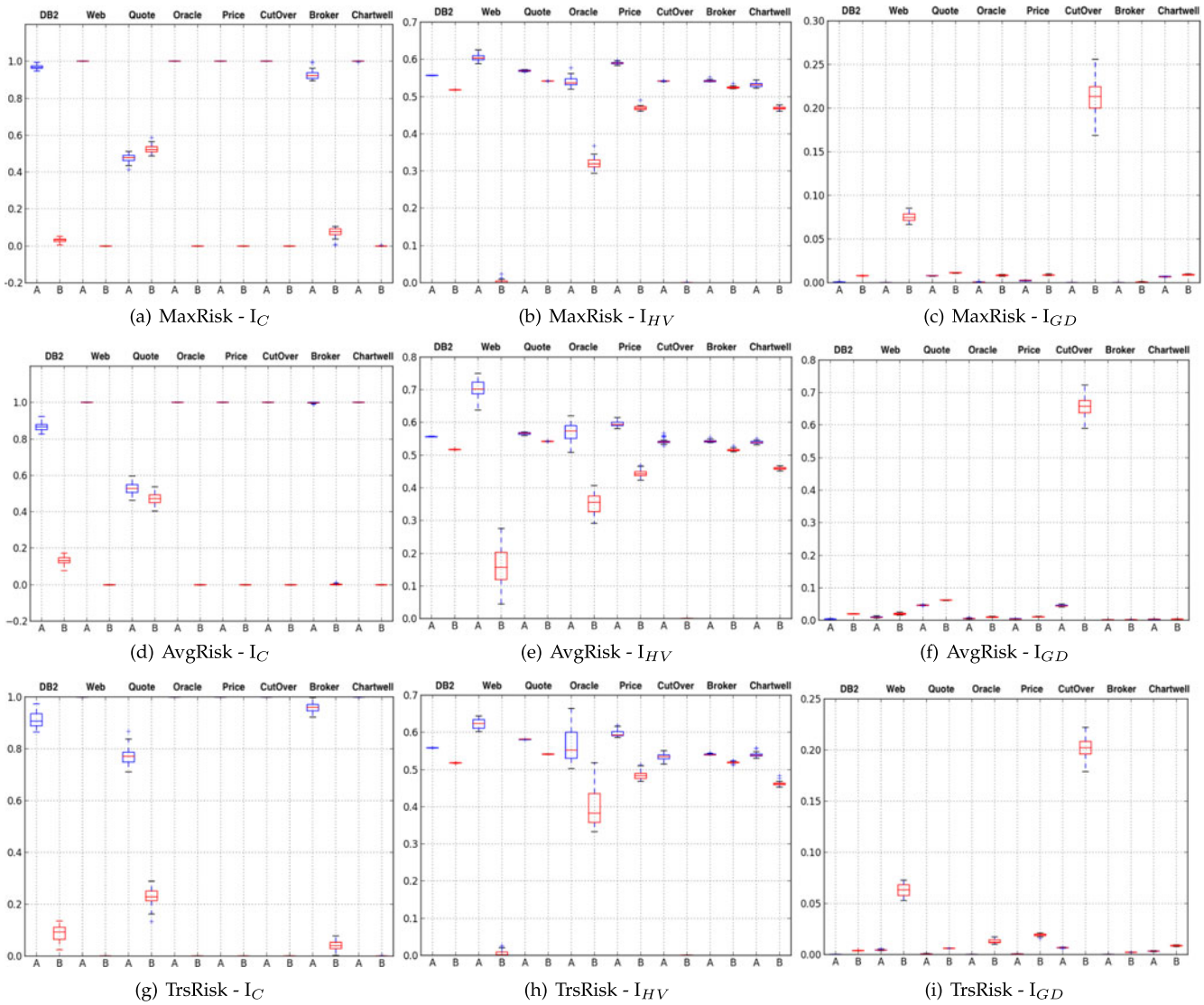


Fig. 2. RQ2: Number of win-tie-loss results from the Wilcoxon Test performed on the three quality indicators (I_c, I_{HV}, I_{GD}) of the pareto fronts obtained by comparing all the NSGAII$_a$ variants for each risk measure on each of the considered datasets.

5.3 Results for RQ3 (Comparison to State of the Art Search)

To assess whether the proposed algorithm (i.e., Adaptive$_{vsc}$) improves the state of the art, we compared it with respect to NSGAII, NSGAII$_v$, and NSGAII$_a$ for the considered datasets.

The boxplots in Fig. 3 show the performance of Adaptive$_{vsc}$ and NSGAII relatively to the three risk measures (i.e., MaxRisk, AvgRisk, TrsRisk) and the three quality assessment indicators (i.e., I_C, I_{HV}, I_{GD}) we considered (see details in Sections 2 and 4.3). We can observe that



(a) MaxRisk - I_C (b) MaxRisk - I_{HV} (c) MaxRisk - I_{GD}

(d) AvgRisk - I_C (e) AvgRisk - I_{HV} (f) AvgRisk - I_{GD}

(g) TrsRisk - I_C (h) TrsRisk - I_{HV} (i) TrsRisk - I_{GD}

Fig. 3. RQ3: Boxplots for the maximal (MaxRisk), average (AvgRisk) and threshold (TrsRisk) risk assessment approaches, evaluated using the quality measures I_C (a), I_{HV} (b), and I_{GD} (c) applied to Adaptive$_{vsc}$ (Algorithm A) and NSGAII (Algorithm B) on each dataset.

Adaptive$_{vsc}$ provides much better results than NSGAII for all the considered datasets. This finding is confirmed by the Wilcoxon test results summarised in Fig. 6a. We observe that the results provided by Adaptive$_{vsc}$ are significantly better than those of the standard NSGAII in 71 out of the 72 experiments (99 percent) with a large $\hat{A}_{12}$ effect size. Only in one case (i.e., project `Quote` for MaxRisk and the I_C quality measure) NSGAII performs better than Adaptive$_{vsc}$ but with a very small effect size (i.e., $\hat{A}_{12} = 0.003$).

Fig. 4 compares the performance of Adaptive$_{vsc}$ and NSGAII$_v$, we can observe that Adaptive$_{vsc}$ provides better results than NSGAII$_v$ in almost all the cases. The Wilcoxon tests confirm this finding (see Fig. 6b): Adaptive$_{vsc}$ significantly outperforms NSGAII$_v$ in 70 out of 72 (97 percent) experiments and in all of these it does so with a large $\hat{A}_{12}$ effect size. In the other two cases no significant difference has been observed.

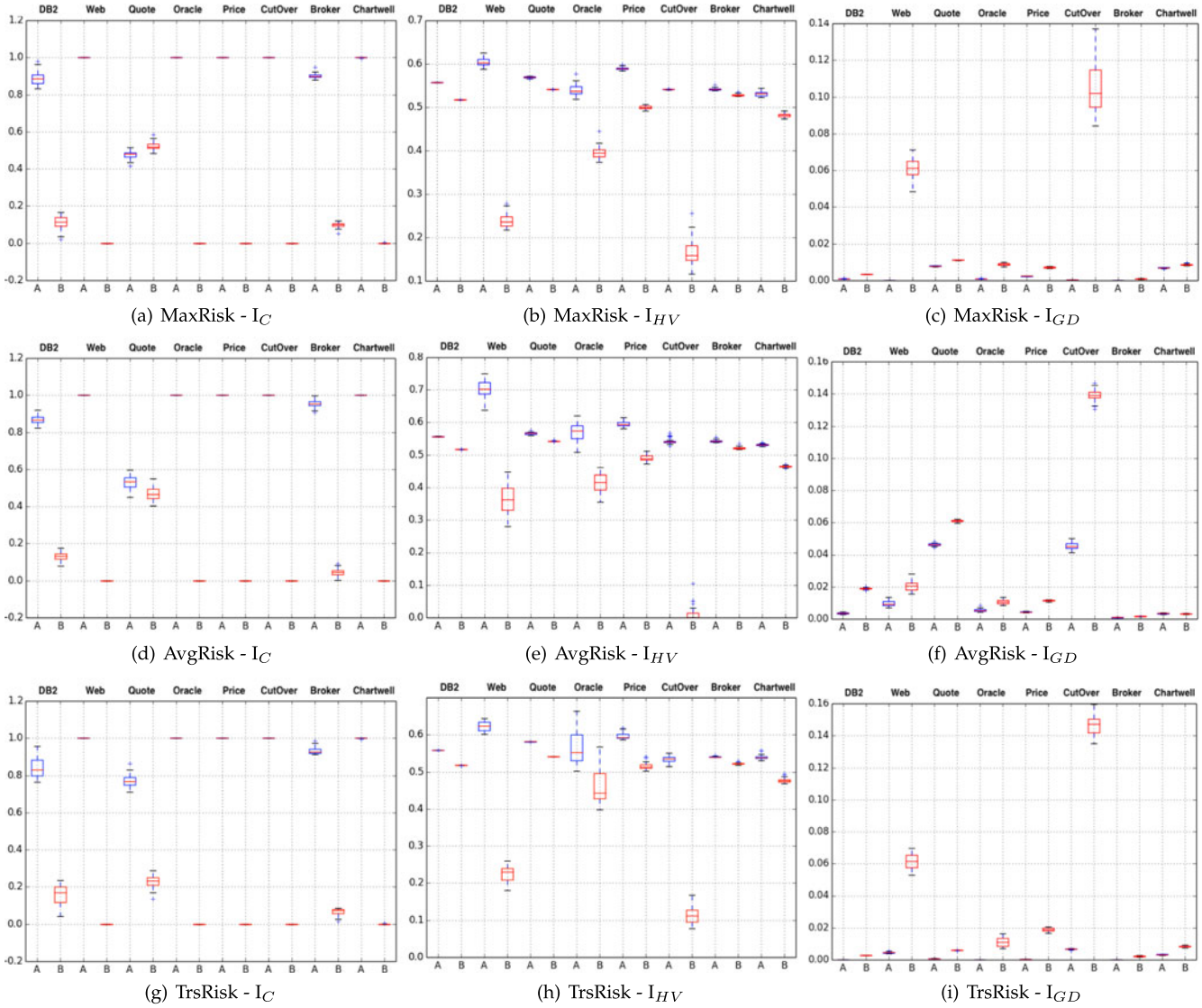
Fig. 5 compares the performance of Adaptive$_{vsc}$ and NSGAII$_a$. We can observe that in most of the cases Adaptive$_{vsc}$ is able to provide better quality indicators than NSGAII$_a$. According to the Wilcoxon Test (see Fig. 6c) Adaptive$_{vsc}$ significantly outperforms NSGAII$_a$ in 59 out of 72 (82 percent) experiments always with a large $\hat{A}_{12}$ effect size. In the remaining 13 cases NSGAII$_a$ performs better than Adaptive$_{vsc}$ with large (eight cases), medium (three cases), and small (two cases) effect sizes.

These results suggest a positive answer to our research question: Adaptive$_{vsc}$ significantly outperforms the state of the art with a large effect size in 202 out of 216 cases (93 percent).

5.4 Results for RQ4 (Usefulness)

In order to answer RQ4, we compared our approach against the 'current overtime planning practice' [11]. There is evidence that current overtime practice employs what has been termed 'margarine management' [11]; spreading the overtime thinly and evenly over all work packages [47]. We can therefore compare our adaptive multi-evolutionary approach to this documented Overtime Management Strategy (OMS).

There are two other natural strategies (often referred to anecdotally in the literature and used in [11]): loading



(a) MaxRisk - I_C (b) MaxRisk - I_{HV} (c) MaxRisk - I_{GD}

(d) AvgRisk - I_C (e) AvgRisk - I_{HV} (f) AvgRisk - I_{GD}

(g) TrsRisk - I_C (h) TrsRisk - I_{HV} (i) TrsRisk - I_{GD}

Fig. 4. RQ3: Boxplots for the maximal (MaxRisk), average (AvgRisk) and threshold (TrsRisk) risk assessment approaches, evaluated using the quality measures I_C (a), I_{HV} (b), and I_{GD} (c) applied to Adaptive$_{vsc}$ (Algorithm A) and NSGAII$_v$ (Algorithm B) on each dataset.

overtime onto the critical path to reduce completion time and loading it onto the later half of the project to compensate for earlier delays.

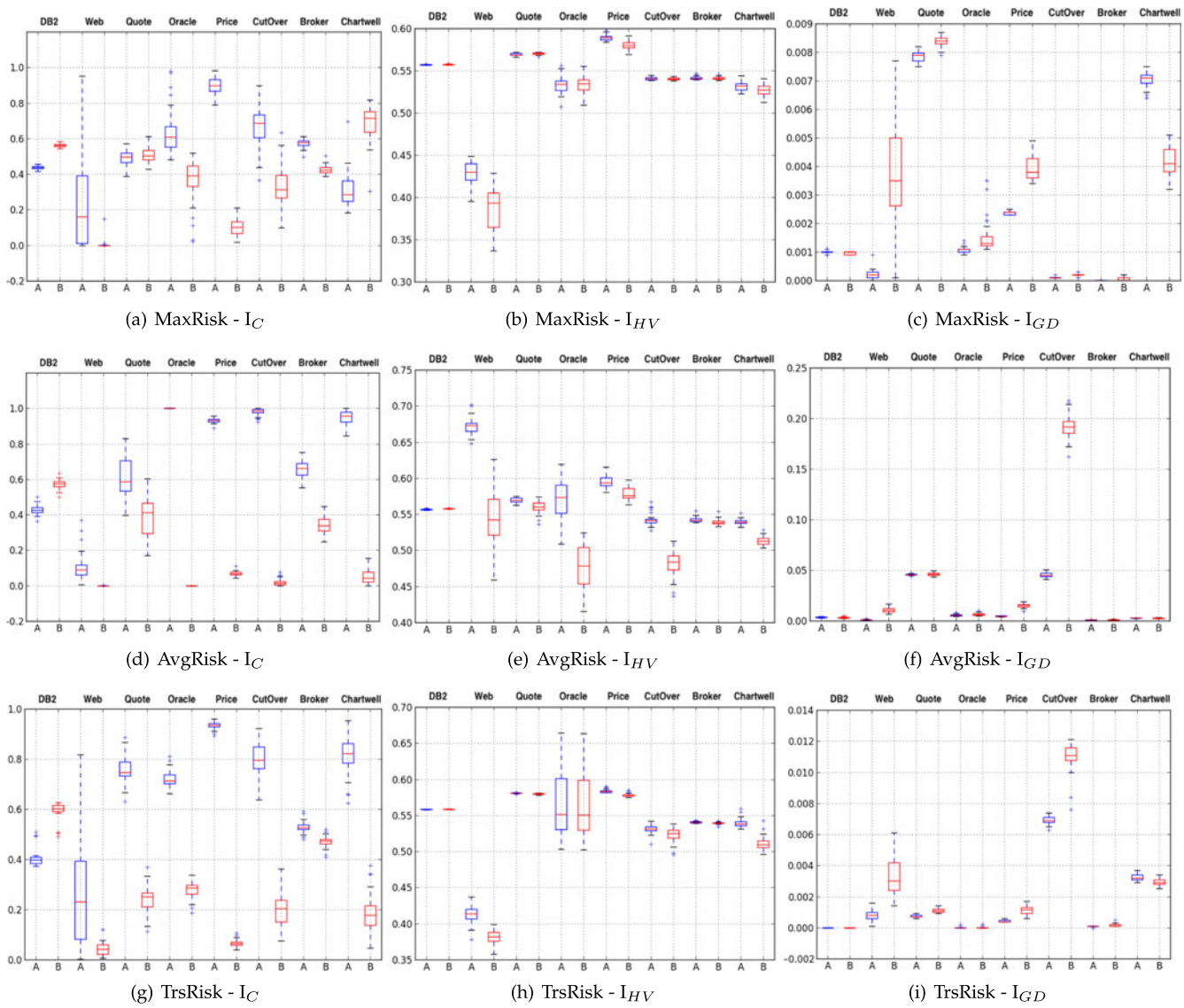
Table 5 reports the mean values of each of the three quality assessment indicators obtained for 30 runs of all projects using Adaptive$_{vsc}$ and the three OMS practices briefly described above. We can observe that Adaptive$_{vsc}$ outperforms these practices. The Wilcoxon Test confirmed that all the indicators obtained by employing Adaptive$_{vsc}$ were significantly better than those obtained with each and all of the OMS practices and with a high $\hat{A}_{12}$ effect size in every case. As an example, Fig. 7 shows the reference fronts obtained by Adaptive$_{vsc}$ and the three OMS practices for the largest project `Web`.[10] For completeness, we also report in Table 6 the mean of the values for each of the objectives provided by the OMS practices and our approach. While Table 5 gives the precise technical answer to RQ4, Fig. 7 provides a more qualitative assessment of the meaning of this technical finding. As can be seen, the Pareto surface produced by Adaptive$_{vsc}$ offers many more points. By contrast, the currently used approaches appear to merely pick relatively arbitrary solutions, which can be sub-optimal (far away from the frontier) and which thus denote little more than rather inaccurate guesses.

10. In this figure overtime is measured in total overtime hours committed to the project, while project duration is measured as the length of the critical path (in days).

6 Threats to Validity

It is widely recognised that several factors can bias the validity of empirical studies. In this section we discuss the validity of our study based on three types of threats, namely *construct*, *internal*, and *external* validity. Construct validity concerns the methodology employed to construct the experiment. Internal validity concerns possible bias in the way in which the results were obtained, while external validity concerns the possible bias of choice of experimental subjects.

In our study, construct validity threats may arise from the assumptions we make about the current state of the



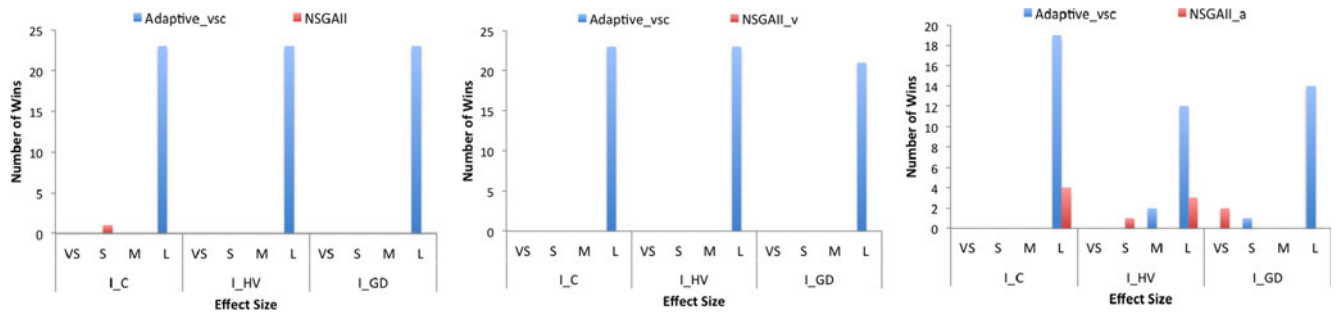
(a) MaxRisk - I_C (b) MaxRisk - I_{HV} (c) MaxRisk - I_{GD}

(d) AvgRisk - I_C (e) AvgRisk - I_{HV} (f) AvgRisk - I_{GD}

(g) TrsRisk - I_C (h) TrsRisk - I_{HV} (i) TrsRisk - I_{GD}

Fig. 5. RQ3: Boxplots for the maximal (MaxRisk), average (AvgRisk) and threshold (TrsRisk) risk assessment approaches, evaluated using the quality measures I_C (a), I_{HV} (b), and I_{GD} (c) applied to Adaptive$_{vsc}$ (Algorithm A) and NSGAII$_a$ (Algorithm B) on each dataset.

art and practice. We found comparatively little literature to guide us on what we should consider to be the 'standard practice' adopted by engineers. We found no new work that addresses the overtime problem since our conference version of the present paper [11], thus we compared against the same three "common current practices' we identified there. As we noted previously [11] there is some degree of support in the literature for one of these choices ('margarine management'), but there is only anecdotal evidence in the literature for the other two practices. Another threat to construct validity can arise from the fact that we did not take into account



(a) Adaptive$_{vsc}$ VS. NSGAII (b) Adaptive$_{vsc}$ VS. NSGAII$_v$ (c) Adaptive$_{vsc}$ VS. NSGAII$_a$

Fig. 6. RQ3: Results of the Wilcoxon Test performed on the Pareto Quality Indicators (i.e., I_c, I_{HV}, I_{GD}) for Adaptive$_{vsc}$ compared to the state of the art (the number of wins obtained by each techniques is grouped per Very Small (VS), Small (S), Medium(M), and Large (L) $\hat{A}_{12}$ effect sizes).

TABLE 5
RQ4: Mean Values of the Quality Indicators for Adaptive$_{vsc}$ and the Three Current Overtime Management Strategies (OMS)

Project	Risk Measure	I_C		I_{HV}		I_{GD}	
		Adaptive$_{vsc}$	OMS	Adaptive$_{vsc}$	OMS	Adaptive$_{vsc}$	OMS
DB2	MaxRisk	0.996	0.003	0.371	0.156	0.003	0.227
	AvgRisk	0.995	0.005	0.653	0.335	0.002	0.059
	TrsRisk	0.994	0.005	0.609	0.000	0.001	0.068
Web	MaxRisk	0.989	0.011	0.209	0.008	0.000	0.555
	AvgRisk	0.994	0.006	0.674	0.225	0.000	0.030
	TrsRisk	0.977	0.023	0.538	0.215	0.001	0.051
Quote	MaxRisk	0.993	0.006	0.408	0.135	0.000	0.247
	AvgRisk	0.993	0.006	0.518	0.230	0.000	0.009
	TrsRisk	0.990	0.009	0.000	0.290	0.000	0.004
Oracle	MaxRisk	0.995	0.005	0.491	0.092	0.000	0.125
	AvgRisk	0.994	0.006	0.641	0.295	0.000	0.019
	TrsRisk	0.994	0.006	0.545	0.360	0.000	0.155
Price	MaxRisk	0.996	0.004	0.528	0.190	0.001	0.103
	AvgRisk	0.997	0.003	0.663	0.365	0.002	0.042
	TrsRisk	0.992	0.008	0.541	0.342	0.000	0.132
CutOver	MaxRisk	0.990	0.009	0.502	0.133	0.012	1.449
	AvgRisk	0.992	0.008	0.723	0.425	0.001	0.006
	TrsRisk	0.982	0.170	0.376	0.365	0.000	0.000
Broker	MaxRisk	0.994	0.006	0.541	0.412	0.000	0.035
	AvgRisk	0.993	0.007	0.554	0.397	0.001	0.021
	TrsRisk	0.993	0.007	0.613	0.431	0.001	0.045
Chartwell	MaxRisk	0.999	0.000	0.206	0.002	0.000	0.047
	AvgRisk	0.991	0.009	0.347	0.117	0.000	0.001
	TrsRisk	0.984	0.016	0.146	0.118	0.002	0.005

resource allocation and skills in the formulation of the problem.

We catered for internal threats to validity in the standard manner for randomised algorithms [21], [39], using non-parametric statistical testing over 30 repeated runs of the algorithms.

Our approach to external threats is also relatively standard for the empirical software engineering literature. That is, while we were able to obtain a set of subjects that had a degree of diversity in scope, application and project team, we cannot claim that our results generalise beyond these subjects studied. The results reported herein use two more datasets, and confirm and extend our original findings [11].

TABLE 6
RQ4. Average Objective Values Achieved by Our Evolutionary Approach (Adaptive$_{vsc}$) and the Three Current Overtime Management Strategies (OMS) for the Project Web

Risk Strategy	Adaptive$_{vsc}$			OMS		
	Overtime	Duration	Risk	Overtime	Duration	Risk
AvgRisk	332.09	92.60	0.28	291.20	100.05	0.28
MaxRisk	101.78	95.93	1.00	11.50	104.56	1.00
TrsRisk	164.61	92.73	0.05	421.67	98.04	0.05

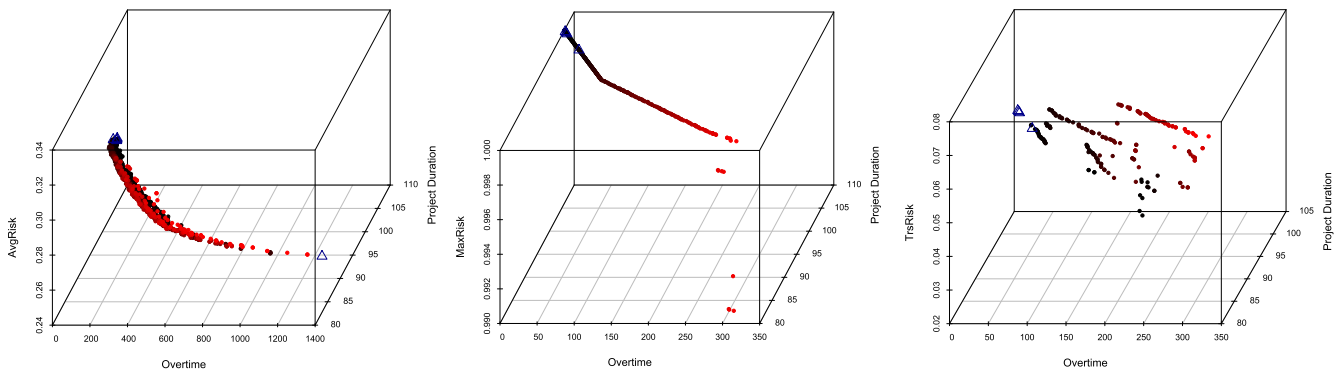
7 Related Work

A comprehensive review on Search Based Project Management can be found elsewhere [45]. In Section 7.1 we summarise the main work in this field by highlighting the difference with the approach we proposed herein, while in Section 7.2 we summarise the main work that analysed the impact of crossover operator and adaptive algorithms in Search Based Software Engineering. The approach used in this paper is also closely related to approaches used in other works on search based software engineering, but *not* (in any way) concerned with project management. We review this closely related work outside the area of Search Based Project Management in Section 7.3.

7.1 Search Based Project Management

For a long time, software engineers have used the Critical Path Method as the principle means of bringing some rudimentary analysis to bear on the problem of project planning [18]. Many software engineers use this approach to plan their projects. However, there have been attempts to replace the human project planner with a more automated planner, based on scheduling and resource optimisation techniques.

The first attempt to apply optimisation to software project planning was the work of Chang et al. [48], who introduced the Software Project Management Net (SPMNet) approach for project scheduling and resource allocation and assesed it on simulated project data. Subsequent research also formulated the problem of constructing an initial project plan as a Search Based Software Engineering problem, using scheduling and simulation [49], [50]. Though most approaches have focused on minimising project duration as



(a) (b) (c)

Fig. 7. RQ4: Pareto surfaces for Adaptive$_{vsc}$ (depicted by the circles) and for all of the three Overtime Management Strategies (depicted by the triangles) obtained using each of the three risk assessment approaches: AvgRisk(a), MaxRisk(b), and TrsRisk(c) for the project Web.

the sole optimisation objective, there has also been work on constructing suitable teams of engineers [51], [52], [53] and work on predicting the effort needed to develop software projects (e.g., [54], [55], [56], [57], [58], [59]).

Previous work has used a variety of SBSE techniques such as Genetic Algorithms [49], Simulated Annealing [51], Co-evolution [31], and Scatter Search [60] as well as hybrids, for example, combining SBSE with constraint satisfaction [61]. Though most of the previous work has been single objective, there has been previous work on multi-objective formulations [51], [62], [63], [64]. However, unlike the present paper, none of this previous work has considered overtime, and all previous work starts with the assumption that it is the role of the optimisation tool (not the software engineer) to provide the initial project plan.

We believe that the assumption that *any* automated tool should have the role of producing the initial project plan, may not always be realistic. Our experience with practitioners is that they would prefer to trust in their own judgement for the initial project plan. This is because the allocation of staff to teams and teams to work packages involves all sorts of human and domain specific judgements for which an automated approach is ill-equipped and a human may be far more suitable.

By contrast, our approach to the overtime planning problem has a fundamentally different starting point and usage scenario in mind: We do not seek to replace the software engineer, nor to second guess their decisions. Rather, we seek to provide decision support in analysing the effects and trade offs in overtime planning. Few software engineers set out with the intention of coercing their team into unplanned overtime, but many well-intentioned and professional software engineers end up doing just that [2], [5]. We seek to provide decision support so that this can be properly planned and better informed by multi-objective risk analysis.

Other authors have considered overtime planning issues in software projects, though none has offered an approach to plan overtime, balancing overtime deployment against project risks. For example, Jia et al. [65] analysed the use of System Dynamics Modeling [66], reporting results on a simulation carried out on a real software project (i.e., ISAM3.1 at Alcatel Shanghai Bell). They report on the harmful effects of excessive overtime (above set limits). Lipke [67] presented a brief report of an effort to control the use of reserve budget in a software project for the defence industry. Barros and Araujo Jr [70] have recently reported some lessons learned by considering both the positive effects of overtime on productivity and its negative effects on product quality. There are many authors who opine overtime's severe negative impacts on staff and their projects (e.g., [6], [7], [8], [68], [69]) but we are the first to offer a technique for automated decision support to help the engineer better plan the deployment of overtime [11]. Moreover, unlike previous work (e.g., [49], [68], [69]), because the approach we use starts with the software engineer's original project plan (rather than attempting to construct it), it requires no simulation, thereby removing this source of potential error and the assumptions that go with it.

7.2 Genetic Operators and Adaptive Algorithms in Search Based Software Engineering

Previous work has discussed the importance of the genetic operators for evolutionary algorithms and adaptive search applied to different optimisation problems (e.g., [71], [72], [73], [74], [75]). In the following we focus our attention on Search Based Software Engineering problems.

Raiha et al. [76] investigated the impact of using crossover for genetically synthesizing software architecture design. They found that although sexual reproduction is favoured among various species of animals and plants, asexual reproduction is more "natural" in the case of genetic synthesis of software architecture. Subsequently, Raiha et al. [77] showed that complementary crossover can significantly improve the use of genetic algorithms to synthesize software architecture.

More recently, Guizzo et al. [78] introduced a metamodel and a mutation operator to allow the application of design patterns in Search Based Product Line Architecture design. The model represents suitable scopes, that is, set of architectural elements that are suitable to receive a pattern. The mutation operator is used with a multi-objective and evolutionary approach to obtain PLA alternatives.

Wang et al. [79] investigated the use of a Memetic Algorithm (MA), based on two genetic operators (i.e., breadth-first crossover and breadth-first mutation) and local search, to maximize the reliabilty of a system by means of Multi-Level Redundancy Allocation. The results showed that the proposed MA significantly outperformed the state of the art approach on two representative examples.

Harman et al. [80] proposed a specific crossover operator for Search Based Optimization of Software Modularization that allowed them to improve the performance of Genetic Algorithm (GA).

Conrad et al. [81] presented a genetic algorithm-based test prioritization method that employs a wide variety of mutation, crossover, selection, and transformation operators to reorder a test suite. The results of their empirical study highlighted the crucial role that the selection operators play in identifying an effective ordering of a test suite.

McMinn analysed how program structure impacts the effectiveness of the crossover operator in evolutionary test input generation and the type of crossover which works most efficiently for different program structures [82], [83]. Harman and McMinn [84] provided evidence that evolutionary testing performs well for Royal Road functions and that this is due to the effect of the crossover operation. Lehre et al. [85] investigated the impact of using crossover on the execution time for the conformance testing of finite state machines. Arcuri and Fraser [86] empirically investigated how GA parameter tuning (e.g., crossover and mutation rate) can have an impact on the performance of the algorithm. The results showed that tuning does indeed have impact but, at least in the context of test data generation, it does not seem easy to find settings that significantly outperform the 'default' values suggested in the literature. Recenlty, LeGoues et al. [87] analysed specific crossover and mutation operators for improving Evolutionary Software Repair. Other work mined and analysed large corpus of software projects to provide useful insights for improving the genetic operators used for automatic program repair [88], [89].

In our previous work [11] we investigated the use of a specific crossover operator to multi-objective overtime management. In the present work we further analysed the use of this crossover together with a multi-objective algorithm able to adaptively use different crossovers during the search. To the best of our knowledge this is the first use of adaptive multi-objective evolutionary algorithms to Search-Based Project Management.

The novel algorithms introduced in the present paper extend our previous work on hyperheuristic learning. This was proposed in 2012 as a means of increase adaptiveness in SBSE [26], but the first results for hyperheuristic SBSE have only recently began to emerge [90].

7.3 Closely Related Work Outside the Area of Search Based Project Management

This paper is concerned with search based project management, which is a subarea of Search Based Software Engineering. One of the advantages of SBSE is the way in which it has the ability to connect apparently unrelated areas of software engineering. Because SBSE solutions can share representations and fitness functions, they can exploit problem formulations that are similar, even though they attack entirely different software engineering application areas. Such connections have been demonstrated, for example, between requirements and regression testing which, from an optimisation perspective, both involve prioritization and selection problems [30]. The work reported in this paper therefore has potential application beyond search based project management, to search based software engineering in general, and to wider search based optimisation problems (that may not even involve software engineering). Evidence for this potential application comes from the way in which other authors have reused the formulation of our evaluation methodology in their own work on SBSE and other multi-objective optimisation problems. For example, Nejati and Briand [91] re-used our evaluation methodology (introduced in our ICSE paper [11] for which the present paper is an extension) in their work on trading CPU and temporal properties, while Olaechea [92] also reused our methodology in their work on multi-objective software product line optimisation. This reuse of the evaluation methodology is not confined to software engineering alone: Ficco et al. re-used our evaluation methodology in their work on optimal selection of positioning systems [93].

8 Conclusions and Future Work

We have extended the search based approach to overtime planning for software engineering projects we proposed in previous work [11] and evaluated it on eight real world software engineering projects. Our approach, evaluated in terms of three standard measures of result quality, performed significantly better (with large effect size) than currently used software engineering practice. Furthermore the adaptive multi-objective evolutionary algorithm introduced in the present work outperformed the state of the art multi-objective algorithms applied to the same problem in 202 out of 216 (93 percent) experiments (with large effect size) showing that the criteria used to adaptively select the crossover during the search together with the use of a crossover specifically designed to the problem in hand allows us to significantly improve the evolutionary algorithm performance.

We provide qualitative evidence that the approach can provide actionable insights to the software engineer, backing up this quantitative evidence that it is effective and useful [11]. As we show in the results presented in our conference paper [11], which are confirmed herein), there exist inflection points that mark sharp differences in the trade-off between additional overtime, and the advantages that accrue from its deployment. These trade-offs cannot be understood without some form of algorithmic approach, since a human cannot be expected to discover such inflection points, unaided.

We believe that this paper lays a firm foundation for future development of semi-automated decision support for software engineers faced with the challenges of planning overtime on complex and demanding projects. However, there remains much to be done to realise the practical benefits that this approach offers.

In future work we plan to deploy a version of the tooling reported upon in this paper as a freely available, open source plug-in component to popular project planning tools, such as Microsoft project. This will allow more extensive evaluation of the interface between the technical aspects of the work reported in this paper and other related socio-technical issues for implementation and exploitation, such as user interface, HCI, and decision support. Moreover, this will allow us also to get feedback from practitioners on the usefulness of the insights provided by our approach and the considered overrun risk strategies. We also plan to collect more data to analyse how well the model performs by applying it on actual projects and comparing the outcomes with projects that use the traditional rule-of-thumb strategies.

Furthermore, it would be interesting to extend the problem formulation considering other aspects such as human and skills allocation [94], team efficiency [95], and voluntary overtime [1] to better represent real world projects and to offer a stronger decision support for software engineers.

Recent results showed that integer linear programming can be successfully applied to the Next Release Problem (NRP) [96] and future work might investigate exact algorithms for project over time planning. Indeed, finding an exact solution would be clearly attractive where possible. However, while it is interesting that there are exact solutions to some problems in search based software engineering, there is no guarantee that the same approach will perform well for a different problem.

Acknowledgments

The authors thank Andreas Andreou, Guenther Ruhe, and Costantinos Stylianou for providing two of the datasets used in the present study. This research was funded by the EPSRC grant EP/J017515 and the Microsoft Azure Research Grant F. Sarro 2014.

Appendix

Table 7 shows the best configuration obtained for each algorithm, per risk measures and per datasets, as a result of the tuning process described in Section 4.5. We used these configurations to answer the research questions set out in Section 4.1.

TABLE 7
Best Obtained Configurations

Project	Technique	MaxRisk	AvgRisk	TrsRisk
DB2	NSGAII	M	VL	L
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	VL	VL	VL
	$Adaptive_s$	VL	VL	VL
	$Adaptive_c$	VL	VL	VL
	$Adaptive_{sc}$	VL	VL	VL
	$Adaptive_{vs}$	VL	VL	VL
	$Adaptive_{vc}$	VL	VL	VL
	$Adaptive_{vsc}$	VL	VL	VL
Web	NSGAII	S	VL	M
	$NSGAII_v$	S	VL	M
	$NSGAII_a$	S	S	S
	$Adaptive_s$	S	M	S
	$Adaptive_c$	M	VL	M
	$Adaptive_{sc}$	VS	VL	M
	$Adaptive_{vs}$	S	S	S
	$Adaptive_{vc}$	VS	VL	VS
	$Adaptive_{vsc}$	VS	VL	L
Quote	NSGAII	VL	VL	VL
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	VL	VL	VL
	$Adaptive_s$	VL	VL	VL
	$Adaptive_c$	VL	VL	VL
	$Adaptive_{sc}$	VL	VL	VL
	$Adaptive_{vs}$	VL	VL	VL
	$Adaptive_{vc}$	VL	VL	VL
	$Adaptive_{vsc}$	VL	VL	VL
Oracle	NSGAII	VL	VL	VL
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	L	VL	L
	$Adaptive_s$	VS	L	M
	$Adaptive_c$	VL	VL	L
	$Adaptive_{sc}$	VL	VL	VS
	$Adaptive_{vs}$	L	L	L
	$Adaptive_{vc}$	VL	VL	VL
	$Adaptive_{vsc}$	VL	VL	VL
Price	NSGAII	VL	VL	VL
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	L	M	M
	$Adaptive_s$	L	M	M
	$Adaptive_c$	VL	VL	M
	$Adaptive_{sc}$	VL	VL	M
	$Adaptive_{vs}$	L	L	M
	$Adaptive_{vc}$	VL	VL	L
	$Adaptive_{vsc}$	VL	VL	VL
CutOver	NSGAII	VL	VL	VL
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	L	M	M
	$Adaptive_s$	VS	M	M
	$Adaptive_c$	L	VL	VS
	$Adaptive_{sc}$	VL	VL	L
	$Adaptive_{vs}$	L	VS	M
	$Adaptive_{vc}$	VS	VL	L
	$Adaptive_{vsc}$	VS	VL	L
Broker	NSGAII	VL	VL	VL
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	VL	VL	VL
	$Adaptive_s$	VL	VL	VL
	$Adaptive_c$	VL	VL	VL
	$Adaptive_{sc}$	VL	VL	VL
	$Adaptive_{vs}$	VL	VL	VL
	$Adaptive_{vc}$	VL	VL	VL
	$Adaptive_{vsc}$	VL	VL	VL
Chartwell	NSGAII	VL	VL	VL
	$NSGAII_v$	VL	VL	VL
	$NSGAII_a$	VL	VL	VL
	$Adaptive_s$	VL	VL	VL
	$Adaptive_c$	VL	VL	VL
	$Adaptive_{sc}$	VL	VL	VL
	$Adaptive_{vs}$	VL	VL	VL
	$Adaptive_{vc}$	VL	VL	VL
	$Adaptive_{vsc}$	VL	VL	VL

REFERENCES

[1] S. McConnel, *Rapid Development: Taming Wild Software Schedules*. Redmond, WA, USA: Microsoft Press, 2010.

[2] M. Nishikitani, M. Nakao, K. Karita, K. Nomura, and E. Yano, "Influence of overtime work, sleep duration, and perceived job characteristics on the physical and mental status of software engineers," *Ind. Health*, vol. 43, no. 4, pp. 623–629, 2005.

[3] B. Akula and J. Cusick, "Impact of overtime and stress on software quality," presented at the 4th Int. Symp. Manag. Eng. Informat., Orlando, FL, USA, 2008.

[4] B. Biafore, *Successful Project Management*. Redmond, WA, USA: Microsoft Press, 2011.

[5] E. Yourdon, *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*. Upper Saddle River, NJ, USA: Prentice-Hall, 1997.

[6] K. Driesen, N. W. H. Jansen, I. Kant, D. C. L. Mohren, and L. G. P. M. van Amelsvoort, "Depressed mood in the working population: Associations with work schedules and working hours," *Chronobiology Int.*, vol. 27, no. 5, pp. 1062–1079, 2010.

[7] E. Kleppa, B. Sanne, and G. S. Tell, "Working overtime is associated with anxiety and depression: The Hordaland health study," *J. Occupational Environmental Med.*, vol. 50, no. 6, pp. 658–666, 2008.

[8] M. Van Der Hulst and S. Geurts, "Associations between overtime and psychological health in high and low reward jobs," *Work Stress: An Int. J. Work Health Organisations*, vol. 15, no. 3, pp. 227–240, 2001.

[9] C. Mann and F. Maurer, "A case study on the impact of scrum on overtime and customer satisfaction," in *Proc. IEEE Comput. Soc. Agile Develop. Conf.*, 2005, pp. 70–79.

[10] D. G. Beckers, D. van der Linden, P. G. Smulders, M. A. Kompier, T. W. Taris, and S. A. Geurts, "Voluntary or involuntary? control over overtime and rewards for overtime in relation to fatigue and work satisfaction," *Work Stress: Int. J. Work Health Organisations*, vol. 22, no. 1, pp. 33–50, 2008.

[11] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, "Not going to take this anymore: Multi-objective overtime planning for software engineering projects," in *Proc. 2013 Int. Conf. Softw. Eng.*, 2013, pp. 462–471.

[12] M. Harman, "The current state and future of search based software engineering," in *Proc. Future Softw. Eng.*, 2007, pp. 342–357.

[13] G. Gay, et al., "Finding robust solutions in requirements models," *Autom. Softw. Eng.*, vol. 17, no. 1, pp. 87–116, Mar. 2010.

[14] M. O. Saliu and G. Ruhe, "Bi-objective release planning for evolving software systems," in *Proc. 6th Joint Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, Sep. 2007, pp. 105–114.

[15] Y. Zhang, A. Finkelstein, and M. Harman, "Search based requirements optimisation: Existing work and challenges," in *Proc. Int. Work. Conf. Requirements Eng.: Found. Softw. Quality*, 2008, pp. 88–94.

[16] A. Nebro, J. Durillo, M. Machn, C. Coello Coello, and B. Dorronsoro, "A study of the combination of variation operators in the NSGA-II algorithm," in *Advances in Artificial Intelligence*. Berlin, Germany: Springer, 2013, pp. 269–278.

[17] C. Chatfield and T. Johnson, *Microsoft Project 2013 Step by Step*. Redmond, WA, USA: Microsoft Press, 2013.

[18] J. E. Kelley, "The critical path method: Resource planning and scheduling," in *Industrial Scheduling*. Upper Saddle River, NJ, USA: Prentice Hall, 1963, pp. 347–365.

[19] P. M. Institute, *A Guide To The Project Management Body Of Knowledge*, 5th ed. Newtown Square, PA, USA: Project Manage. Institute, 2013.

[20] F. G. Freitas and J. T. Souza, "Ten years of search based software engineering: A bibliometric analysis," in *Proc. 3rd Int. Symp. Search Based Softw. Eng.*, 2011, pp. 18–32.

[21] M. Harman, P. McMinn, J. Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical Software Engineering and Verification: LASER 2009–2010*, B. Meyer and M. Nordio, Eds. Berlin, Germany: Springer, 2012, pp. 1–59.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[23] K. De Jong, "On using genetic algorithms to search program spaces," in *Proc. 2nd Int. Conf. Genetic Algorithms Their Appl.*, Jul. 28–31, 1987, pp. 210–216.

[24] M. Harman and P. McMinn, "A theoretical and empirical study of search based testing: Local, global and hybrid search," *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 226–247, Mar. 2010.

[25] M. Harman, R. Hierons, and M. Proctor, "A new representation and crossover operator for search-based optimization of software modularization," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 9–13, 2002, pp. 1351–1358.

[26] M. Harman, E. Burke, J. A. Clark, and X. Yao, "Dynamic adaptive search based software engineering," in *Proc. 6th IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Sep. 2012, pp. 1–8.

[27] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

[28] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances Eng. Softw.*, vol. 42, pp. 760–771, 2011.

[29] G. Toscano Pulido and C. A. Coello Coello, "The micro genetic algorithm 2: Towards online adaptation in evolutionary multiobjective optimization" in *Proc. 2nd Int. Conf. Evol. Multi-Criterion Optim.*, 2003, pp. 252–266.

[30] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering: Trends, techniques and applications," *ACM Comput. Surveys*, 2012, vol. 45, no. 1, Art. no. 11.

[31] J. Ren, M. Harman, and M. Di Penta, "Cooperative co-evolutionary optimization on software project staff assignments and job scheduling," in *Proc. 3rd Int. Symp. Search Based Softw. Eng.*, Sept. 10–11, 2011, pp. 127–141.

[32] P. Kapur, A. Ngo-The, G. Ruhe, and A. Smith, "Optimized staffing for product releases and its application at chartwell technology," *J. Softw. Maintenance Evol.: Res. Practice*, vol. 20, no. 5, pp. 365–386, 2008.

[33] C. Stylianou, S. Gerasimou, and A. Andreou, "A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors," in *Proc. IEEE 24th Int. Conf. Tools Artif. Intell.*, Nov. 2012, pp. 277–284.

[34] J. D. Knowles, L. Thiele, and E. Zitzler, "A tutorial on the performance assessment of stochastic multiobjective optimizers," Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland, Tech. Rep. 214, 2006.

[35] H. Meunier, E.-G. Talbi, and P. Reininger, "A multiobjective genetic algorithm for radio network optimization," in *Proc. Congr. Evol. Comput.*, 2000, pp. 317–324.

[36] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.

[37] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 117–132, Apr. 2003.

[38] D. A. V. Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm research: A history and analysis," Department of Electrical and Computer Engineering, Wright-Patterson AFB, Ohio, Tech. Rep. TR-98-03, 1998.

[39] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 1–10.

[40] P. Royston, "An extension of Shapiro and Wilk's W test for normality to large samples," *Appl. Statist.*, vol. 31, no. 2, pp. 115–124, 1982.

[41] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Denmark: Lawrence Earlbaum Associates, 1988.

[42] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of mcgraw and wong," *J. Edu. Behavioral Statist.*, vol. 25, no. 2, pp. 101–132, jan 2000.

[43] G. Neumann, M. Harman, and S. Poulding, "Transformed Vargha-Delaney effect size," in *Proc. 7th Int. Symp. Search-Based Softw. Eng.*, 2015, pp. 318–324.

[44] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Inform. Softw. Technol.*, vol. 48, no. 11, pp. 1034–1045, 2006.

[45] F. Ferrucci, M. Harman, and F. Sarro, "Search-based software project management," in *Software Project Management in a Changing World*, G. Ruhe and C. Wohlin, Eds. Berlin, Germany: Springer, 2014, pp. 373–399.

[46] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1403–1416, Nov./Dec. 2012.

[47] A. S. Hanna and G. Haddadl, "Overtime and productivity in -electrical construction," in *Proc. Construct. Res. Congr.*, 2009, pp. 1–10.

[48] C. K. Chang, C. Chao, S.-Y. Hsieh, and Y. Alsalqan, "Spmnet: A formal methodology for software management," in *Proc. 18th Annu. Int. Comput. Softw. Appl. Conf.*, Nov. 9–11, 1994, pp. 57–57.

[49] E. Alba and F. Chicano, "Software project management with gas," *Inform. Sci.*, vol. 177, no. 11, pp. 2380–2401, Jun. 2007.

[50] G. Antoniol, M. D. Penta, and M. Harman, "Search-based techniques applied to optimization of project planning for a massive maintenance project," in *Proc. 21st IEEE Int. Conf. Softw. Maintenance*, 2005, pp. 240–249.

[51] G. Antoniol, M. Di Penta, and M. Harman, "The use of search-based optimization techniques to schedule and staff software projects: An approach and an empirical study," *Softw.: Practice Experience*, vol. 41, no. 5, pp. 495–519, Apr. 2011.

[52] M. Hericko, A. Zivkovic, and I. Rozman, "An approach to optimizing software development team size," *Inform. Process. Lett.*, vol. 108, no. 3, pp. 101–106, Oct. 2008.

[53] D. Kang, J. Jung, and D.-H. Bae, "Constraint-based human resource allocation in software projects," *Softw.: Practice Experience*, vol. 41, no. 5, pp. 551–577, Apr. 2011.

[54] M. Lefley and M. J. Shepperd, "Using genetic programming to improve software effort estimation based on general data sets," in *Proc. Genetic and Evol. Comput. Conf.*, 2003, pp. 2477–2487.

[55] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, *Using Tabu Search to Estimate Software Development Effort*. Berlin, Germany: Springer, 2009, pp. 307–320.

[56] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Genetic programming for effort estimation: An analysis of the impact of different fitness functions," in *Proc. 2nd Int. Symp. Search Based Softw. Eng.*, 2010, pp. 89–98.

[57] F. Sarro, F. Ferrucci, and C. Gravino, "Single and multi objective genetic programming for software development effort estimation," in *Proc. 27th Annu. ACM Symp. Appl. Comput.*, 2012, pp. 1221–1226.

[58] A. Corazza, S. D. Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using tabu search to configure support vector regression for effort estimation," *Empirical Softw. Eng.*, vol. 18, no. 3, pp. 506–546, 2013.

[59] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective effort estimation," in *Proc. 38th Int. Conf. Softw. Eng. Res. Track*, 2016, pp. 619–630.

[60] R. Alvarez-Valds, E. Crespo, J. M. Tamarit, and F. Villa, "A scatter search algorithm for project scheduling under partially renewable resources," *J. Heuristics*, vol. 12, no. 1–2, pp. 95–113, Mar. 2006.

[61] A. Barreto, M. de Oliveira Barros, and C. M. L. Werner, "Staffing a software project: A constraint satisfaction and optimization-based approach," *Comput. Operations Res.*, vol. 35, no. 10, pp. 3073–3089, Oct. 2008.

[62] F. Chicano, F. Luna, A. J. Nebro, and E. Alba, "Using multi-objective metaheuristics to solve the software project scheduling problem," in *Proc. 13th Annu. Conf. Genetic Evol. Comput.*, Jul. 12–16, 2011, pp. 1915–1922.

[63] G. Antoniol, S. Gueorguiev, and M. Harman, "Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering," in *ACM Proc. Genetic Evolutionary Comput. Conf.*, 2009, pp. 1673–1680.

[64] F. R. Burton, R. F. Paige, S. M. Poulding, and S. Smith, "System of systems acquisition trade-offs," in *Proc. Conf. Syst. Eng. Res.*, 2014, pp. 11–18.

[65] J. Jia, X. Fan, and Y. Lu, "System dynamics modeling for overtime management strategy of software project," in *Proc. 25th Int. Conf. Syst. Dynamics Soc.*, 2007, pp. 1–8.

[66] J. M. Lyneisa and D. N. Fordb, "System dynamics applied to project management: A survey, assessment, and directions for future research," *Syst. Dynamics Rev.*, vol. 23, no. 2–3, pp. 157–189, 2007.

[67] W. H. Lipke, "Applying management reserve to software project management," *Crosstalk: The J. Defense Softw. Eng.*, vol. 3, pp. 17–21, Mar. 1999.

[68] L. L. Minku, D. Sudholt, and X. Yao, "Evolutionary algorithms for the project scheduling problem: Runtime analysis and improved design," in *Proc. Genetic Evol. Comput. Conf.*, 2012, pp. 1221–1228.

[69] L. Minku, D. Sudholt, and X. Yao, "Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis," *IEEE Trans. Softw. Eng.*, vol. 40, no. 1, pp. 83–102, Jan. 2014.

[70] M. d. O. Barros and L. A. O. d. Araujo, Jr., "Learning overtime dynamics through multiobjective optimization," in *Proc. Genetic and Evol. Comput. Conf.*, 2016, pp. 1061–1068.

[71] W. Spears and V. Anand, "A study of crossover operators in genetic programming," in *Methodologies for Intelligent Systems*, Z. Ras and M. Zemankova, Eds. Berlin, Germany: Springer, 1991, vol. 542, pp. 409–418.

[72] P. C. Pendharkar and J. A. Rodger, "An empirical study of impact of crossover operators on the performance of non-binary genetic algorithm based neural approaches for classification," *Comput. Oper. Res.*, vol. 31, no. 4, pp. 481–498, Apr. 2004.
[73] D. R. White and S. Poulding, "A rigorous evaluation of crossover and mutation in genetic programming," in *Proc. 12th Eur. Conf. Genetic Program.*, Apr. 15–17, 2009, pp. 220–231.
[74] M. Harman, E. Burke, J. Clark, and X. Yao, "Dynamic adaptive search based software engineering," in *Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2012, pp. 1–8.
[75] P. Consoli, L. Minku, and X. Yao, "Dynamic selection of evolutionary algorithm operators based on online learning and fitness landscape metrics," in *Simulated Evolution and Learning*. Berlin, Germany: Springer Int. Pub., 2014, pp. 359–370.
[76] O. Raiha, K. Koskimies, and E. Makinen, "Empirical study on the effect of crossover in genetic software architecture synthesis," in *Proc. 1st World Congr. Nature Biologically Inspired Comput.*, Dec. 9–11, 2009, pp. 619–625.
[77] O. Raiha, K. Koskimies, and E. Mkinen, "Complementary crossover for genetic software architecture synthesis," in *Proc. 10th Int. Conf. Intell. Syst. Des. Appl.*, Nov. 29-Dec. 1 2010, pp. 266–271.
[78] G. Guizzo, T. E. Colanzi, and S. R. Vergilio, "A pattern-driven mutation operator for search-based product line architecture design," in *Proc. 6th Int. Symp. Search-Based Softw. Eng.*, 2014, pp. 77–91.
[79] Z. Wang, K. Tang, and X. Yao, "A memetic algorithm for multi-level redundancy allocation," *IEEE Trans. Rel.*, vol. 59, no. 4, pp. 754–765, Dec. 2010.
[80] M. Harman, R. Hierons, and M. Proctor, "A new representation and crossover operator for search-based optimization of software modularization," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 9–13, 2002, pp. 1351–1358.
[81] A. P. Conrad, R. S. Roos, and G. M. Kapfhammer, "Empirically studying the role of selection operators during search-based test suite prioritization," in *Proc. 12th Annu. Conf. Genetic Evol. Comput.*, 2010, pp. 1373–1380.
[82] P. McMinn, "How does program structure impact the effectiveness of the crossover operator in evolutionary testing?" in *Proc. 2nd Int. Symp. Search Based Softw. Eng.*, 2010, pp. 9–18.
[83] P. Mcminn, "An identification of program factors that impact crossover performance in evolutionary test input generation for the branch coverage of C programs," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 153–172, Jan. 2013.
[84] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 226–247, Mar.-Apr. 2010.
[85] P. K. Lehre and X. Yao, "Crossover can be constructive when computing unique input-output sequences," *Soft Comput.—A Fusion Found. Methodologies Appl.*, vol. 15, no. 9, pp. 1675–1687, 2011.
[86] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Softw. Eng.*, vol. 18, no. 3, pp. 594–623, 2013.
[87] C. Le Goues, W. Weimer, and S. Forrest, "Representations and operators for improving evolutionary software repair," in *Proc. 14th Annu. Conf. Genetic and Evol. Comput.*, 2012, pp. 959–966.
[88] E. T. Barr, Y. Brun, P. T. Devanbu, M. Harman, and F. Sarro, "The plastic surgery hypothesis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Foundations Softw. Eng., Nov.*, 2014, pp. 306–317.
[89] H. Zhong and Z. Su, "An empirical study on fixing real bugs," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 913–923.
[90] Y. Jia, M. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction test generation strategies using hyperheuristic search," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 540–550.
[91] S. Nejati and L. C. Briand, "Identifying optimal trade-offs between cpu time usage and temporal constraints using search," in *Proc. 2014 Int. Symp. Softw. Testing Anal.*, 2014, pp. 351–361.
[92] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki, "Comparison of exact and approximate multi-objective optimization for software product lines," in *Proc. 18th Int. Softw. Product Line Conf.*, 2014, pp. 92–101.
[93] M. Ficco, R. Pietrantuono, and S. Russo, "Using multi-objective metaheuristics for the optimal selection of positioning systems," *Soft Comput.*, vol. 20, pp. 2641–2664, 2016.
[94] C. Stylianou and A. Andreou, "Human resource allocation and scheduling for software project management," in *Software Project Management in a Changing World*, G. Ruhe and C. Wohlin, Eds. Berlin, Germany: Springer, 2014, pp. 73–106.
[95] N. Jin and X. Yao, "Heuristic optimization for software project management with impacts of team efficiency," in *Proc. IEEE Congr. Evol. Comput.*, 2014, pp. 3016–3023.
[96] N. Veerapen, G. Ochoa, M. Harman, and E. K. Burke, "An integer linear programming approach to the single and bi-objective next release problem," *Inform. Softw. Technol.*, vol. 65, pp. 1–13, 2015.

Federica Sarro is a senior research associate in the Department of Computer Science at University College London, where she is member of the CREST centre. Her main research areas are empirical software engineering and search based software engineering (SBSE), with a particular interest in software effort estimation, defect prediction, project planning, and app stores mining and analysis. She has published more than 50 papers in top SE venues and won 4 international awards including the HUMIES award at GECCO 2016. She has served on more than 40 program committees and as the program co-chair for SBSE at GECCO 2017 and SSBSE 2016, and been elected onto the steering committee for SSBSE in 2015. She has also been guest editor for the *IEEE Transactions on Evolutionary Computation* journal and member of the editorial board of the *Springer Genetic Programming and Evolvable Machines* journal. Web page: http://www0.cs.ucl.ac.uk/staff/F.Sarro/.

Filomena Ferrucci is professor of software engineering and software project management at University of Salerno, Italy. Her main research interests include software metrics, effort estimation, search-based software engineering, empirical software engineering, and human-computer interaction. She has been program co-chair of the International Summer School on Software Engineering. Web page: http://docenti.unisa.it/001775/home.

Mark Harman is professor of Software Engineering in the Department of Computer Science at University College London, where he directs the CREST centre and is Head of Software Systems Engineering. He is widely known for work on source code analysis, software testing, app store analysis and Search Based Software Engineering (SBSE), a field he co-founded and which has grown rapidly to include more than 1,600 authors spread over more than 40 countries. His SBSE and testing work has been used by many organisations including Daimler, Ericsson, Google, Huawei, Microsoft and Visa. He is co-director of Appredict, an app store analytics company, spun out from UCL's UCLappA group, and chief scientific advisor to Majicke, and automated test data generation start up. Web page: http://www0.cs.ucl.ac.uk/staff/mharman/.

Alessandra Manna received the MSc degree in computer science from University of Salerno, Italy, in 2014. She was also an Erasmus student at University College London working for 6 months in the CREST research centre on the use of evolutionary approaches for software project management. Successively, she worked on geomapping and visualization algorithms at the University of Innsbruck. She currently works in Innsbruck as a Software Engineer for a company providing cochlear implants.

Jian Ren received the two MSc degrees from Queen Mary University of London and Kings College London, respectively, and the PhD degree in computer science from the University College London, in 2013. He is an assistant professor in the School of Computer Science at Beihang University, Beijing. His research interests include search based software engineering, software project planning and management, requirements engineering, and evolutionary computation.