Improving Multi-Objective Test Case Selection by Injecting Diversity in Genetic Algorithms

Annibale Panichella, *Member, IEEE*, Rocco Oliveto, *Member, IEEE*,
Massimiliano Di Penta, *Member, IEEE*, and Andrea De Lucia, *Senior Member, IEEE*

Abstract—A way to reduce the cost of regression testing consists of selecting or prioritizing subsets of test cases from a test suite according to some criteria. Besides greedy algorithms, cost cognizant additional greedy algorithms, multi-objective optimization algorithms, and multi-objective genetic algorithms (MOGAs), have also been proposed to tackle this problem. However, previous studies have shown that there is no clear winner between greedy and MOGAs, and that their combination does not necessarily produce better results. In this paper we show that the optimality of MOGAs can be significantly improved by diversifying the solutions (sub-sets of the test suite) generated during the search process. Specifically, we introduce a new MOGA, coined as DIversity based Genetic Algorithm (DIV-GA), based on the mechanisms of orthogonal design and orthogonal evolution that increase diversity by injecting new orthogonal individuals during the search process. Results of an empirical study conducted on eleven programs show that DIV-GA outperforms both greedy algorithms and the traditional MOGAs from the optimality point of view. Moreover, the solutions (sub-sets of the test suite) provided by DIV-GA are able to detect more faults than the other algorithms, while keeping the same test execution cost.

Index Terms—Test case selection, regression testing, orthogonal design, singular value decomposition, genetic algorithms, empirical studies

1 Introduction

Regression testing consists of re-testing software that has been modified. Such an activity is required to verify whether new changes have introduced errors into unchanged parts, endangering their behaviors [69]. Re-testing the whole software system by executing all the available test cases might be too expensive and unfeasible, especially for large systems [25], [55]. Running some test suites can take hours, even days, so developers cannot exercise the system instantly or in reasonable time [58]. The problem is clearly amplified by the growth of the test suites as the system evolves.

Several strategies have been proposed to reduce the effort of regression testing by selecting a (possibly minimal) subset of test cases from the test suite with respect to some testing criteria [5], [6], [14], [24], [29], [37], [48], [50], [52], [54], [57], [71], [73], or prioritizing their execution with the purpose of first executing those believed to reveal faults earlier [20], [21], [44], [66], [71]. In general, solving these problems requires the tester (i) choosing some testing criteria to be satisfied, and (ii) using an optimization technique (e.g., greedy or search-based algorithm) to select/order the test cases on the basis of the chosen criteria. For example, widely-used criteria are code coverage [6], [21], [29], program modification [24], [52], [61], execution cost [20], [44], [71], or past fault information [6], [71], [72].

The problem of test suite optimization has been also formulated as a combination of multiple—often contrasting—criteria. Results have highlighted that the optimization of a test suite is more effective when using multiple criteria than when using individual ones [6], [37], [59], [71], [72]. The simplest way to combine different criteria is to conflate all the criteria in a single-objective function to be optimized [20], [21], [29], [44]. Although such an approach is widely used when solving multi-objective optimization problems, this may produce less optimal results compared to Pareto-efficient methods. Thus, Yoo and Harman [71], [72] treated the test suite optimization problems using Pareto-efficient multi-objective genetic algorithms (MOGAs) to deal with multiple and contrasting objectives. Empirical results indicated that in some cases MOGAs provide better solutions. However, there is no a clear winner between single-objective greedy algorithms and MOGAs [71] and their combination is not always useful to achieve better results [72].

We conjecture that a reason for the poor performances exhibited by MOGAs in several cases as compared to single-objective algorithms is represented by the phenomenon of the *genetic drift*, i.e., a loss of diversity in the Genetic Algorithm (GA) population [34]. In essence, while search-based optimization techniques can be useful for regression test selection, usually such techniques only try to find (near) optimal solution with respect to some fitness functions. They do not guarantee the diversity of the produced solutions. Looking at multi-objective optimization techniques, this lack of diversity often leads to stagnation because MOGAs generate offspring not diversified enough with respect to their parents, lacking the genetic diversity needed to escape from local regions [65]. In such a scenario MOGAs

- *A. Panichella and A. De Lucia are with the Department of Mathematics and Computer Science, University of Salerno, Fisciano 84084, Salerno, Italy. E-mail: {apanichella, adelucia}@unisa.it.*
- *R. Oliveto is with the Department of Bioscience and Territory, University of Molise, Pesche 86090, Isernia, Italy. E-mail: rocco.oliveto@unimol.it.*
- *M. Di Penta is with the Department of Engineering, University of Sannio, Benevento 82100, Italy. E-mail: dipenta@unisannio.it.*

Manuscript received 17 Nov. 2013; revised 31 July 2014; accepted 26 Sept. 2014. Date of publication 26 Oct. 2014; date of current version 17 Apr. 2015. Recommended for acceptance by M. Pezze.
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.2014.2364175

can prematurely converge within some sub-optimal regions [1], [15], [34], [40], [45].

We also conjecture that, even if standard MOGAs include mechanisms aimed at maintaining diversity between solutions in the *phenotype* space (i.e., with respect to the objective scores of these solutions) [65], such mechanisms do not adequately prevent genetic drift in the context of test suite optimization. The empirical evidence of this conjecture can be derived by the results of previous empirical studies [41], [71], [72], which demonstrated that these mechanisms do not always help in outperforming simple greedy algorithms, because MOGAs converged prematurely to some sub-optimal solutions.

Some approaches for test suite optimization also attempt to inject diversity in the phenotype space. Hemmati et al. [30], [31] exploit test case diversity (based on UML state machine coverage) as a unique test criterion to be optimized when selecting test cases. The idea is that higher diversity between test cases in the solution provided by the algorithm mirrors higher coverage. Unlike approaches based on MOGAs this single-objective approach is not able to provide tradeoffs between diversity and test execution cost. To overcome this problem, in a previous study [14] we suggested adding a diversity-preserving objective function (measured according to a coverage criterion, such as code coverage) to typical multi-objective formulations aimed at minimizing the cost and maximizing the coverage. The additional objective function promotes diversity between the solutions (sub-sets of the test suite) of the population of MOGAs, rather than between the test cases of a solution, like in the approach by Hemmati et al. [30], [31]. However, different coverage criteria might require different diversity-based objective functions, one function for each coverage criterion. Since the performance of MOGAs rapidly decreases for an increasing number of objective functions [39], it is preferable to promote diversity without adding further objective functions.

Stemming from these considerations, in this paper we introduce two novel genetic operators to promote diversity between the candidate solutions (sub-sets of the test suite) in the *genotype space* rather than in the *phenotype space*. Specifically, we introduce (i) a generative algorithm to build a diversified initial population, based on *orthogonal design* [49], and (ii) an orthogonal exploration mechanism of the search space, through singular value decomposition (SVD) [63], aimed at preserving the diversity during the evolution of the population [15]. Since these two mechanisms are defined in the genotype space, they can be applied for any test suite optimization problem and independently of the number of test criteria or objectives to be taken into account. It is worth noting that, while diversity in the phenotype space naturally implies diversity in the genotype space, it is not necessarily true the opposite. However, as our goal is to avoid genetic drift, injecting diversity in the genotype space increases the probability of avoiding being trapped in local optima and then to get more quickly to global optima. Therefore, our conjecture is that promoting diversity in the genotype space through orthogonal exploration of the search space also results in diversity in the phenotype space.

We conducted an empirical study on eleven real world open-source programs. The results show that there is a strong relationship between genotype and phenotype diversities, since when promoting the genotype diversity we also improve phenotype diversity. We also find that DIVersity based Genetic Algorithm (DIV-GA) outperforms both traditional MOGAs and greedy algorithms. Unlike previous work on multi-criteria regression test case selection (TCS) [14], [71], [72], which compared meta-heuristics and greedy algorithms only from an optimization point of view, in this paper we also present a performance metric to evaluate the ability of the selected test cases to reveal faults (effectiveness) in a multi-objective paradigm. In particular, we defined a novel metric to measure the cost-effectiveness of a test suite that is inspired by the traditional hypervolume metric widely used for numeric multi-objective problems [4]. The experimental results reveal the superiority of DIV-GA as compared with traditional MOGAs and greedy algorithms also in terms of cost-effectiveness.

Summarizing, the contributions of this paper are:

1) a new MOGA, called DIVersity based Genetic Algorithm (DIV-GA) which integrates orthogonal evolution and orthogonal design into MOGAs to solve multi-criteria test case selection problems. DIV-GA addresses the problem of diversity in the genotype space, thus, independently of the number and the kind of test criteria;
2) the evaluation of DIV-GA on a set of open-source programs from the Siemens suite, the European Space Agency suite and GNU open-source distribution. The selected programs were also used in many previous work [9], [14], [37], [56], [57], [67], [71], [72], [73].
3) The comparison of DIV-GA with previous techniques, namely greedy algorithms and the island version of NSGA-II [17] (named vNSGA-II), previously used by Yoo and Harman for test suite optimization [14], [71], [72], [73]. The comparison concerns both optimality and effectiveness.

The paper is organized as follows. After a discussion of the related literature (Section 2), Section 3 presents the orthogonal design and the SVD-based orthogonal evolution, and describes how to integrate such operators into the main loop of MOGAs. Section 4 describes the design of the empirical study we conducted to evaluate the benefits of the proposed algorithm. Results are reported and discussed in Section 5, while Sections 6 and 7 provide additional empirical analyses and discuss the threats that could affect the validity of the results of our study, respectively. Section 8 concludes the paper.

2 Background and Related Work

Approaches aimed at reducing the effort of regression testing include *test suite minimization (TSM)* (or reduction), *test case selection*, and *test case prioritization (TCP)*. A complete survey of such approaches can be found in the paper by Yoo and Harman [69]. The following sections provide a discussion of the most relevant related work. In particular, Section 2.1 provides an overview on traditional approaches to test suite optimization, while Section 2.2 focuses on search-based approaches. Finally, Section 2.3 describes the main diversity-preserving mechanisms used in literature.

2.1 Test Suite Optimization: An Overview of Traditional Approaches

The goal of the test suite minimization problem consists of reducing the size of the test suite by deleting test cases that are redundant with respect to some coverage criteria [57], such as code coverage, branch coverage, data flow, dynamic program invariants or call stacks [29]. Clearly, one issue of the test suite minimization is that removing some test cases from the test suite may potentially affect its ability to detect faults, since a smaller test suite might have a lower effectiveness [56], [67]. Finding the minimal subset of a test suite is NP-complete, as it can be reduced to the *minimal hitting set* problem in polynomial time. Hence, several heuristics have been applied to deal with this problem [9], [29], [50]. Harrold et al. [29] used the greedy algorithm for the minimal hitting set problem, while Chen and Lau [9] used it for solving its dual problem, i.e., the set covering problem. Offutt et al. [50] used greedy approaches with different test case ordering criteria instead of the fixed ordering of the original one. An empirical comparison of greedy approaches suggested that none of them is able to outperform the others [50]. Further work based on greedy approaches considered other coverage criteria than the code-level structural coverage criteria used by Harrold et al. [29], Offutt et al. [50], and Chen and Lau [9]. For example, Marré and Bertolino [46] formulated the TSM problem as the problem of finding a minimal spanning set over the *decision-to-decision* graph. McMaster and Memon [48] proposed a test suite minimization technique based on call-stack coverage. Black et al. [6] considered a bi-criteria approach that takes into account two testing criteria: (i) code coverage and (ii) past fault detection history. They combined the two objectives by applying a weighted-sum approach, and used integer linear programming (ILP) optimization to find subsets, then reducing the multi-objective problem to a single-objective one.

Test case prioritization is aimed at ordering test cases to maximize some desired properties [53]. It involves the execution of the test cases in a given order and terminating the testing process at some arbitrary point chosen by the decision maker [69]. The ideal ordering of test cases is the one that maximizes the actual fault detection rate, but it is only known after test execution. Hence, the ordering criterion generally depends on surrogates which are in some way correlated with the fault detection rate, such as code coverage [20], [23], [53], interaction coverage [7], clustering-based coverage [12], and requirement coverage [60]. According to the chosen surrogate, the ordering of the test cases is computed using a greedy algorithm, since the ordering by which the test cases are selected by the algorithm also mirrors the ordering to execute them. Greedy algorithms have also been used to solve a bi-criteria TCP problem by conflating two objectives (coverage and cost) in only one function (coverage per unit cost) to be maximized by applying the weighted-sum approach [20], [44]. Rothermel et al. [53] provided empirical evidence of the usefulness of the prioritization techniques with respect to the random ordering by measuring the ability to early detect faults. A similar analysis was performed by Do et al. [18] using the Java unit test framework (JUnit). Elbaum et al. [22] considered further testing criteria with different granularity, e.g., statement coverage or function coverage.

Test case selection focuses on selecting a subset from an initial test suite to test software changes, i.e., to test whether unmodified parts of a program still continue to work correctly after changes involving other parts [54]. The identification of the modified parts of software can be performed using different techniques, including Integer Programming [24], symbolic execution [68], data flow analysis [52], dependence graph based techniques [5], and flow graph-based approaches [54]. The details of the different selecting approaches differ based on how a specific technique defines, seeks and identifies changes in the program under test [69]. Once the test cases covering the unmodified parts of programs are identified using a given technique, an optimization algorithm—e.g., additional greedy—can be used to select a minimal set of such test cases according to some testing criteria—e.g., statement coverage—with the purpose of reducing the cost of regression testing.

As it has been previously pointed out by Yoo and Harman [71], [72], test suite minimization, test case selection and test case prioritization are strongly related to each others. For example, both test suite minimization and test case selection involve the selection of elements (test cases) from a test suite (starting set) that best satisfy the testing criteria (e.g., code coverage) [28]. Test case prioritization is also highly related to test case selection [61], [71], [72], since an optimal ordering can be applied to the test cases aiming at reducing the cost of regression testing. For instance, Srivastava and Thiagarajan [61] combine prioritization and test case selection. Specifically, they first detect the unmodified parts of software by comparing the binary code before and after changes. Hence, a greedy algorithm is used to order test cases but only according to the coverage of unmodified parts.

2.2 Search-Based Test Suite Optimization

Test case selection, test suite minimization, and test case prioritization can be viewed as multi-objective problems, were the goal is to select a Pareto-efficient subset of the test suite, based on multiple test criteria [71], [72]. Multi-objective algorithms, such as MOGAs, can then be applied to solve them. A complete analysis of the benefits of this multi-objective reformulation is also provided by Sampath et al. [59] showing that the combined (hybrid) test criterion often outperformed their constituent individual criteria.

Let $\Gamma = \{\tau_1, \ldots, \tau_n\}$ be a test suite and $F = \{f_1, \ldots, f_m\}$ a set of objective functions, i.e., the mathematical descriptions of test criteria to be satisfied during the selection of test cases. The multi-objective test suite selection problem considered in this paper can be defined as follows [71]: selecting a subset $\Gamma' \subseteq \Gamma$ such that Γ' is the Pareto-optimal set with respect to the objective functions in F. The optimality of the solutions is measured through the concepts of Pareto optimality and Pareto dominance. It is important to note that this search-based formulation is referred to the test case selection problem tackled in this paper, which does not require test case ordering. A different formulation taking into account test case ordering—i.e., using tuples instead of sets—is required for the test case prioritization problem.

A solution X is said to be *Pareto-optimal* if and only if it is non-dominated by any other solution within the search space, i.e., if and only if no other solution Y exists which would improve one of the objective functions, without

worsening other objectives. All the solutions that are not dominated by any other solution are said to form a *Pareto-optimal set*, while the corresponding *objective vectors* (containing the values of the objective functions) are said to form a *Pareto frontier*. Identifying a Pareto frontier is particularly useful because the software engineer can use the frontier to make a well-informed decision that balances the trade-offs between the different objectives. For example, the software engineer can choose the solution with lower execution cost or higher code coverage on the basis of the resources available for executing the selected test cases.

Yoo and Harman [71], [72] considered two and three contrasting test criteria: code coverage and execution time in the two-objective formulation; then, they added the fault history information as third criterion in the three-objective formulation. They also evaluated different optimization algorithms to find Pareto-optimal sub-sets of the test suite: additional greedy algorithms and a variant of the multi-objective genetic algorithm NSGA-II [17]. The additional greedy algorithms were applied by using the traditional weighted sum approach to conflate all the objectives in only one function to be optimized: a cost cognizant version of the additional greedy algorithm was used for the two-objective formulation, while the weighted sum of code coverage per unit of time and fault coverage per unit of time was considered for the three-objective formulation. The empirical comparison between MOGAs and greedy algorithms did not reveal a clear winner between them, and in some cases the MOGAs were not able to outperform the greedy algorithms [71]. Furthermore, the combination between these two kinds of algorithms was not always useful to reach better results [72]. Also, greedy algorithms—which perform well for single-objective formulations—are not always Pareto-efficient in the multi-objective paradigm, motivating the use of meta-heuristic techniques [71], [72]. Similar considerations have also been provided by Li et al. [41], who investigated many meta-heuristic algorithms for the single-objective formulation, including hill climbing algorithms, GAs, additional greedy algorithms, and two-optimal greedy algorithms.

2.3 Avoiding Genetic Drifts through Diversity

In this paper we conjecture that the cause of the poor performance of MOGAs is represented by the phenomenon of *genetic drift*, i.e., a loss of diversity in the population [34], [65], which can lead towards a premature convergence within some sub-optimal region. Indeed, after few generations, a GA tends to create few groups of solutions (niches), that are close to each other in the search space, leaving the rest of the search space unexplored. In general, the problem of maintaining diversity during the search process has been recognized as a crucial problem for ensuring the optimality of GAs, especially for multi-objective optimization problems [19]. The importance of diversity is demonstrated by the large amount of work aimed at addressing such an issue for numerical problems [15], [17], [33], [42], [74], [75]. A complete classification of the diversity-preserving mechanisms proposed in literature to prevent genetic drift when solving numerical problems can be found in a recent survey by Črepinšek et al. [65].

According to this survey, diversity mechanisms in multi-objective optimization try to improve the diversity between solution in the *phenotype* space (for test case selection higher phenotype diversity indicates that different sub-test suites have different objective scores) by acting on different components of the GA, i.e., (i) by modifying the fitness function, (ii) by adding new similarity-based fitness functions, and (iii) by promoting diversity during the selection process. For example, widely used diversity-preserving mechanisms are *fitness sharing* [26], [34], *crowding distance* [43], *restricted tournament selection* [27], *rank scaling selection* [31].

The island variant of the NSGA-II algorithm used by Yoo and Harman [71], [72] (named vNSGA-II) is of particular interest in this context because it includes three different diversity-preserving mechanisms for promoting phenotype diversity between candidate solutions (set of selected test cases): *crowding distance*, *ranking based selection* and *migration between sub-populations* (islands). In particular, NSGA-II [17] uses the concept of *crowding distance* to decide which solutions have to be selected for the next generation (in this case the diversity is promoted during the selection process). The crowding distance measures how far a solution is from the rest of the population [16]—according to the objective functions—and then giving to diversified solution a higher probability of survival. From the test suite optimization point of view, this means that given a particular coverage criterion, the crowding distance will promote set of test cases that are diversified according to the corresponding coverage criterion. NSGA-II also uses a second diversity mechanism since it selects individuals on the basis of a rank scaling function where the rank is measured according to the *non-dominated sorting algorithm*. Finally, in addition to crowding distance and ranking based selection, the vNSGA-II algorithm used by Yoo and Harman [71], [72] uses separated sub-populations instead of a single population, in order to achieve wider Pareto frontiers. Periodically, vNSGA-II introduces diversity by exchanging individuals between sub-populations with the idea that different sub-populations can maintain different promising regions of the search space [65]. Previous work by Yoo and Harman [71], [72], also demonstrated the superiority of vNSGA-II against the standard NSGA-II. However, diversity-preserving mechanisms do not always help NSGA-II and its island version in outperforming simple greedy algorithms in the context of test case selection [72]. Thus, further diversity mechanisms are required to achieve better results.

For numeric problems, another important way to maintain diversity is to consider diversity as an explicit further objective function in a multi-objective paradigm [64], [65], where diversity can be measured using a specific distance function (e.g., euclidean distance) between solutions in the genotype, phenotype or according to external criterion. In a previous work [14] we investigated the possibility of enhancing the two-objective formulation of the test case selection problem by adding a third objective function promoting diversity in the phenotype space. In particular, a density function has been added, with the aim of ensuring diversity between the individuals of the population according to the statement coverage criterion. Results indicated the usefulness of diversity to improve multi-objective GAs.

The approach proposed by Hemmati et al. [30] also follows this direction, but considers diversity as the unique objective function to be optimized, where diversity is measured by using a similarity function between pairs of test cases, according to a coverage criterion on a test model [32]. Hemmati et al. also provided a set of strategies to select test cases such as adaptive random testing (ART) [8] and single-objective GAs. Among the different selection strategies, GAs turned out to be the most effective technique for *similarity-based test case selection* [30]. Further studies [31], [32] also confirmed that the diversity increases the ability to detect faults and evolutionary algorithms turned out to be most efficient for such an objective function.

The main disadvantage of these approaches that consider diversity as an explicit objective function [14], [30], [31], [32] is that such an objective is represented by a density/distance function specific for a given coverage criterion. In a multi-objective paradigm where multiple coverage criteria can be used, such approaches require the addition of a diversity function for each coverage criterion, thus increasing the number of objective functions to be (near) optimized. As noted by Köppen and Yoshida [39], the performance of MOGAs rapidly decreases for an increasing number of objective functions. Thus, it is preferable to promote diversity without adding further objective functions, while acting on other steps of the evolution process, such as the selection mechanism or the generation of new individuals.

This is the main goal of our paper. Specifically, we suggest to promote diversity between solutions (candidate sub-test suites) in the genotype space: higher genotype diversity indicates that different solutions select different sets of test cases. Then, we introduce two genetic operators to this aim: (i) the *orthogonal design* [49] to build a diversified initial population [75], and (ii) an orthogonal exploration mechanism of the search space [15], through singular value decomposition [63]. In particular, the *orthogonal design* acts in the initialization of the MOGAs, while the orthogonal exploration acts during the evolution by injecting individuals that, according to the SVD, are diversified enough. Both genetic operators are independent of he number of testing criteria to be considered, and they do not require the addition of further diversity-aware criteria. Details are provided in Section 3.

Previous studies used a simple binary coding representation of solutions for multi-objective test suite optimization, where the *i-th* digit of the binary string is 1 if the test case τ_i is included in the solution and 0 otherwise. Recently, Yoo [70] pointed out that the widely used binary string representation may not be ideal when using coverage criteria for test case selection and, thus, he proposed a novel and different representation of solutions, called *mask-coding*. An empirical study demonstrated that the performances of MOGAs can depend on the solution representation and in some case can improve both the optimality and phenotype diversity between the obtained solutions. However, as also explained by Yoo [70], in some cases solutions obtained by the *mask-coding* are worse than the ones obtained with traditional binary-coding. In this paper we focus on the traditional binary-coding for two main reasons: (i) there is no clear winner between mask-coding and binary-coding; (ii) the binary coding is the most used; and (iii) the binary-coding is also the only one that has been compared with greedy algorithms.

3 Injecting Diversity in Multi-Objective Genetic Algorithms: DIV-GA

This section describes how we use DIV-GA to solve the multi-objective test case selection problem. Specifically, we detail how we inject diversity into the main loop of NSGA-II, which is the Pareto efficient multi-objective genetic algorithm designed by Deb et al. [17]. While previous approaches to multi-objective test case selection [14], [71], [72] used the island variant of NSGA-II (vNSGA-II), we based DIV-GA on the standard version NSGA-II. As explained later in this Section, the orthogonal exploration mechanism of the search space through singular value decomposition works on a unique population, which is incompatible with the sub-populations used by vNSGA-II.

As any other GA, NSGA-II uses multiple solutions or individuals, also called *chromosomes*, which are evolved in parallel to explore different parts of the search space. As shown in Algorithm 1, NSGA-II starts with an initial set of random solutions (random sub-test suites in our case) called a *population*, obtained by randomly sampling the search space (line 3 of Algorithm 1). The population then evolves through a series of iterations, called *generations* to find nearby better solutions. To produce the next generation, NSGA-II first creates new individuals, called *offspring*, by merging the genes of two individuals in the current generation using a *crossover* operator or modifying a solution using a *mutation* operator (function MAKE-NEW-POP [17], at line 5 of Algorithm 1).

Algorithm 1. NSGA-II

Input:
A test suite of size N
Population size M
Result: A set of Pareto efficient sub-test suites S

```
1  begin
2    t ⟵ 0 // current generation
3    P_t ⟵ RANDOM-POPULATION(N, M)
4    while not (end condition) do
5      Q_t ⟵ MAKE-NEW-POP(P_t)
6      R_t ⟵ P_t ⋃ Q_t;
7      𝔽 ⟵ FAST-NONDOMINATED-SORT(R_t)
8      P_{t+1} ⟵ ∅
9      d ⟵ 1
10     while |P_{t+1}| + |𝔽_d| ⩽ M do
11         CROWDING-DISTANCE-ASSIGNMENT(𝔽_d)
12         P_{t+1} ⟵ P_{t+1} ⋃ 𝔽_d
13         d ⟵ d + 1
14     Sort(𝔽_d) //according to the crowding distance
15     P_{t+1} ⟵ P_{t+1} ⋃ 𝔽_d[1 : (M − |P_{t+1}|)]
16     t ⟵ t + 1
17   S ⟵ P_t
18 end
```

A new population is generated using a *selection* operator, to select parents and offspring according to the values of the objective functions. The process of selection is performed using the concept of Pareto optimality, which leads the selection of non-dominated solutions in the current population. The *crowding distance* is used in order to make a decision about which individuals to select: the individuals that are far away from the rest of the population have higher

probability to be selected. Furthermore, NSGA-II uses the *fast non-dominated sorting* algorithm to preserve in the next generation the individuals forming the current Pareto frontier (*elitism*). After some generations, the algorithm converges to "stable" solutions, i.e., the Pareto-optimal set of the problem.

In line 7, the function FAST-NON-DOMINATED-SORT [17] assigns the non-dominated ranks to individuals parents and offsprings. The loop between lines 10 and 14 adds as many individuals as possible to the next generation, according to their non-dominance ranks, until reaching the population size. Specifically, the algorithm first selects the non-dominated solutions from the first non-dominated front (F_1); if the number of selected solutions is lower than the population size M, the loop selects the non-dominated solutions from the second non-dominated front (F_2), and so on. The loop will stop when adding the solutions of the current non-dominated front F_d would exceed the population size M. In case the number of selected solutions at the end of the loop is lower than the population size M, the algorithm selects the remaining solutions from the current non-dominated front F_d according to the descending order of crowding distance in lines 14-15.

The next sections describe in detail the two diversity mechanisms proposed in this paper. Specifically, Section 3.1 describes how to generate diversified initial populations [74] using the orthogonal (or experimental) design [49], while Section 3.2 describes the concept of *evolution directions*, and how to estimate such directions to promote diversity [15]. Finally, Section 3.3 explains how to integrate these techniques into the main loop of NSGA-II.

3.1 Diversity in the Initial Population

The function used to generate an initial population plays an important role on the performance of GAs [42] since it performs an initial sampling of the search space. A well-distributed and well-diversified initial population makes the exploration more effective and favors GA convergence toward global optima [42]. Instead, a poorly diversified initial population can compromise the convergence speed and the optimality of the search space. This issue becomes particularly critical for problems where the length of the chromosome (number of test cases in our case) is larger than the size of the population [10]. This is especially true for the test case selection problem, where searching the optimal subset—according to multiple testing criteria—of a test suite of size n requires the analysis of 2^n possible solutions.

According to Zhang and Leung [74], statistical methods such as *orthogonal design* can be used to improve the optimality and the convergence speed of GAs. A generic solution of the test suite minimization problem is an array of binary digits $X = \{x_1, \ldots, x_n\}$ where x_i is equal to 1 if the *i-th* test case is selected, 0 otherwise. Then, each element x_i can be considered as a two-level factor[1] which affects the outcome of the fitness function. Hence, the problem of generating a well-distributed initial population for GAs is equivalent to the problem of finding a (small) representative sample of all the possible combinations between factors (test cases) for a given experiment.

Because of such an equivalence, we propose to use the *orthogonal arrays* methodology designed by Montgomery [49] for orthogonal design, to generate an initial population for GAs. Several numeric algorithms can be used to generate the orthogonal arrays, such as the *row-exchange algorithm* or the *coordinate-exchange algorithm* [49]. In this paper, we use the Hadamard matrices to build the orthogonal arrays, since such a methodology is particularly efficient for generating two-level orthogonal arrays [62], i.e. the ones required for binary problems such as the test case selection problem. By definition, a Hadamard matrix H of order n is an $n \times n$ matrix with the property that:

$$H \times H^T = H^T \times H = nI \tag{1}$$

with all elements either equal to $+1$ or -1, where I is the identity matrix. The meaning of this property is that all the row vectors in H are mutually orthogonal (i.e., they have inner product equals to 0) as well as all the column vectors in H. This also implies that we can select any subset of the rows (or of columns) and still we have a set of mutually orthogonal vectors. Moreover, all the row and column vectors contain the same number of $+1$ and -1 elements with the exception of the first column and the first row. An example of Hadamard matrix of size 4 is the following:

$$H_4 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}.$$

There is more than one technique available to build Hadamard matrices. However, the simplest (and also most powerful) method to build larger Hadamard matrices consists of using the following properties: *if H is a Hadamard matrix of order n, then the following matrix is a Hadamard matrix of size $2n$:*

$$H_{2n} = \begin{pmatrix} H & H \\ H & -H \end{pmatrix}. \tag{2}$$

It has been shown that a Hadamard matrix always has an order n which is a multiple of 4 [62]. In the context of this paper, we are interested in Hadamard matrices because their row vectors (or equivalently their column vectors) represent a special cases of orthogonal arrays (with an order which must be a multiple of 4) with two-level factors $\{-1; +1\}$. In order to generate orthogonal arrays of any size/order and with values in $\{0; 1\}$, we need to manage the Hadamard matrices.

Let N be the size of the test suite and let M be the number of individuals to be generated, we generate the orthogonal arrays (or equivalently the initial population for GAs) using the steps reported in Algorithm 2. Steps at lines 3 and 4 have been implemented using the `hadamard` and the `sortrows` routines, respectively, available in MATLAB [47]. In line 5, we remove the first column because it is the unique one which contains only +1 elements, thus, if we maintain this column then the first test cases will be

1. A two-level factor is a factor assuming only two possible values [49]. In our case $x_i \in \{0, 1\}$.

selected by all the individuals in the initial population (and it will be the only test case with this peculiarity). Since the obtained H matrix will contains $k \geq m$ rows and $k-1 \geq n$ columns, we select the first m rows and the first n column (line 6) in order to have the desired $m \times n$ matrix, whose m rows are orthogonal arrays of length n. It is important to note that we could select any subset of rows or columns in H but the selected ones still need to be mutually orthogonal vectors. Finally, line 7 converts the matrix L with values $\in \{-1, 1\}$ into a new matrix $L_m(2^n)$ with values $\in \{0, 1\}$.

Algorithm 2. ORTHOGONAL-POPULATION

Input:
The size of the test suite n; The size of the initial population m;
Result: An initial population P_0 of m individuals.

1 **begin**
2 $N \longleftarrow \max\{m, n\}$
3 Generate a Hadamard matrix H_k of size $k = \lceil (N+1)/4 \rceil \times 4$, where $\lceil (N+1)/4 \rceil$ denotes the smallest integer number greater or equal to $(N+1)/4$
4 Sort the rows of H_k in ascending order
5 Delete the first column from H_k
6 Select the first m rows and the first n columns from H_k to obtain a new matrix L of size $m \times n$
7 Convert L in a binary matrix $L_m(2^n)$
8 $P_0 \longleftarrow L_m(2^n)$
9 **end**

We propose to use the row vectors of such a matrix $L_m(2^n)$ as individuals of an initial population for GAs, where its generic entry $L_{j,i}$ is equal to 1 if the ith test case is selected by the jth individual, 0 otherwise (step 6 in Algorithm 2). According to Zhu et al. [75], all the properties of the orthogonal arrays make them very suitable to be used as initial population for GAs, since they guarantee the minimal mutual information between individuals and a scattered uniformly sampling of the search space.

Note that the orthogonal arrays used by Zhu et al. [75] have more than two-level factors and have been used to solve real-coded numerical problems, while we use two-level orthogonal arrays since test case selection is a multi-objective problem whose solutions are binary arrays. Moreover, in this paper we suggest to use a generative approach to build the orthogonal arrays based on Hadamard matrices, which is more efficient for binary populations [62] than the iterative algorithm proposed by Zhu et al. [75].

3.2 Diversity during Population Evolution

Even if the initial population is well-diversified, during the evolution process—i.e., across different generations—the diversity between individuals can be compromised with the risk to explore the same search regions while other parts of the search space are left unexplored (premature convergence) [1], [34], [40], [45]. The idea of GAs is that a population tends to evolve toward regions of the search space with better fitness because at each generation a selection operator is used to select the (best) individuals that have to survive in the next generation. For multi-objective problems the *selection operator* selects for reproduction the individuals that are non-dominated by any other solution in the current population, i.e. individuals with the best *compromise* between the contrasting objectives (which represents an approximation of the optimal Pareto set). As new generations are produced, the best individuals will tend to converge towards some locally- or globally-optimal Pareto regions. However, even if the evolution process is based on the randomness nature of mutation and crossover operators, the population can be trapped in some locally-optimal regions (*genetic drift* or *premature convergence* [10]).

Recently, De Lucia et al. [15] observed that it is possible to identify where the best individuals of the population are evolving over two consecutive generations. Such movements, called *evolution directions*, are estimated through singular value decomposition [63], and are used to inject diversity in the population in order to push its evolution toward unexplored/orthogonal regions. This is achieved by replacing the worst part of the population with new individuals that are orthogonal to the best part of the population. This technique has been applied to real-coded numerical problems [15] and also for evolutionary test data generation [38], showing the benefits of SVD-based diversity on the effectiveness and efficiency of GAs.

In this paper we propose to adapt the approach proposed by De Lucia et al. [15] to the test case selection problem. Specifically, the approach by De Lucia et al. [15] has been adapted at three main points:

1) encoding of solution (binary-coded chromosomes instead of real-coded ones);
2) selection of best individuals which is performed using the concept of Pareto optimality (while in the work by De Lucia et al. [15] and by Kifetew et al. [38] the best individuals were selected according to the definition of *fittest* individuals in single-objective paradigm); and
3) once we have generated new individual through SVD, we reconvert the obtained real-coded vectors in binary vectors (this step was not required in the previous work by De Lucia et al. [15] and by Kifetew et al. [38]).

More details about these customizations for the test case selection problem are provided in the following when describing the main steps of the proposed algorithm.

The steps needed to inject diversity during the evolution of the population are reported in Algorithm 3. The algorithm takes as input two populations P_t and P_{t+k} obtained by a multi-objective genetic algorithm at generations t and $t+k$ and produce as output a new population P^* obtained by injecting diversity in population P_{t+k}. Each population can be considered as $m \times n$ matrix where n is the number of test cases, m is the number of individuals, while the generic entry $p_{i,j}$ is equal to 1 if the *j-th* test case is selected by the *i-th* individual.

The first two steps of the algorithm (at lines 2 and 3) select 50 percent of the best individuals from the two input populations, $P_t^{'}$ and P'_{t+k} respectively. The selection of the best 50 percent of individuals is performed using the *fast non-dominated sorting* algorithm and the concept of *crowding distance* [17], i.e., the traditional *operators* used by NSGA-II to assign the ranks (levels of optimality) to all the individuals of a given population.

Algorithm 3. INJECT-DIVERSITY

Input:
A population P_t
A population P_{t+k}
Result: A new population P^*_{t+k}
1 **begin**
2 $P'_t \longleftarrow$ 50% of best individuals of P_t
3 $P'_{t+k} \longleftarrow$ 50% of best individuals of P_{t+k}
4 $[U_t, \Sigma_t, V_t] \longleftarrow svd(P'_t)$
5 $[U_{t+k}, \Sigma_{t+k}, V_{t+k}] \longleftarrow svd(P'_{t+k})$
6 $\overline{V} \longleftarrow V_{t+k} - V_t$ //Evolution directions
7 $\overline{\Sigma} \longleftarrow \Sigma_{t+k} - \Sigma_t$ //Shifting operator
8 //Generate orthogonal evolution directions
9 $\overline{V}^o \longleftarrow$ ORTHOGONAL-DIRECTION$(\overline{V})$
10 //Generate new orthogonal individuals
11 $P^o \longleftarrow U_{t+k} \cdot (\Sigma_{t+k} + \overline{\Sigma}) \cdot (V_{t+k} + \overline{V}^o)^T$
12 Convert the elements in P^o in binary values
13 $P^* \longleftarrow P'_{t+k} \cup P^o$
14 **end**

Then, SVD is applied in steps (lines 4 and 5) to decompose each of these two sub-populations as $P' = \left(U \cdot \Sigma \cdot V^T\right)$ for identifying and ordering the dimensions (axes) along which the individuals exhibit most of the variation. The column vectors of V denote the main directions along which the individuals are distributed in the search space, while the diagonal elements of Σ measure the *importance* of each main direction v_j for the population distribution. By definition such a decomposition also orders the vectors $V = \{v_1, v_2, \ldots, v_n\}$ in descending order of importance. Thus, the individuals exhibit the first largest variation along the direction v_1, the second largest variation along the direction v_2, and so on. Consequently, the diagonal elements $\sigma_{i,i}$ in Σ are also ordered in descending order of magnitude.

Clearly, two populations obtained in two different generations have different SVD factorizations, because they are distributed differently in the search space. Since SVD allows us to capture such distributions, a simple way to estimate where the non-dominated solutions are evolving after k generations consists of comparing the two corresponding decompositions $\{\Sigma_t, V_t\}$ at generation t and $\{\Sigma_{t+k}, V_{t+k}\}$ at generation $t+k$. Specifically, $\overline{V} \longleftarrow V_{t+k} - V_t$ computed at line 6 estimates the *directions* along which the population is evolving (*evolution directions*) and $\overline{\Sigma} \longleftarrow \Sigma_{t+k} - \Sigma_t$ computed at line 7 measures the *magnitude* of its evolution.

Starting from these matrices, it is possible to generate a new set of individuals P^o as reported at line 11 with orthogonal directions as compared to directions of the current population. The matrix $\overline{V}^o$ computed at line 9 is a matrix whose column vectors are unit and orthogonal with respect to column vectors of $\overline{V}$, i.e., $\overline{v}^o_i$ is orthogonal to $\overline{v}_i$ for $i = 1 \ldots n$. Since the number of all possible orthogonal column vectors is infinite, a good choice would be to randomly generate such vectors. We have chosen the simple method shown in Algorithm 4 [38]. Such an algorithm creates orthogonal vectors in a simple way. Given a vector $\overrightarrow{v}_i$, its orthogonal vector can be computed by (i) creating a new vector $\overrightarrow{v}^o_i$ with the same elements in $\overrightarrow{v}_i$ but in reverse order

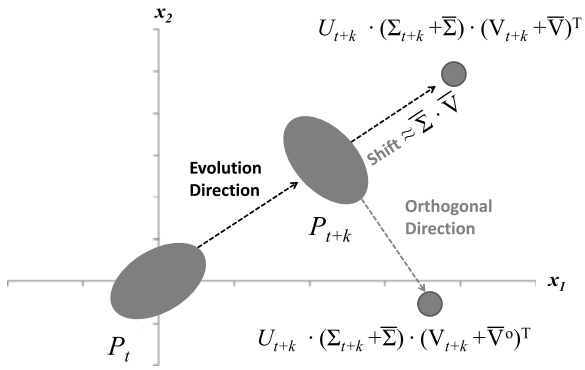


Fig. 1. SVD-based GA: Graphical interpretation of orthogonal diversification.

(line 3 of Algorithm 4), and (ii) randomly multiplying the first or last 50 percent of its elements by -1 (lines 4-8 of Algorithm 4). If the number of elements in $\overrightarrow{v}^o_i$ is odd, then we randomly set to zero one of its unmodified element in order to ensure the orthogonality with $\overrightarrow{v}_i$ (lines 9-10 of Algorithm 4).

Algorithm 4. ORTHOGONAL-DIRECTION

Input:
A $m \times n$ matrix $\overline{V}$;
Result: A matrix $\overline{V}^o$;
1 **begin**
2 **foreach** Column $i \in \{1, \ldots, n\}$ **do**
3 $\overrightarrow{v}^o_i \longleftarrow$ reverse-order$(\overrightarrow{v}_i)$
4 $\eta \longleftarrow$ random number ranging in $[0; 1]$
5 **if** $\eta > 0.5$ **then**
6 multiply by -1 the first $\lfloor m/2 \rfloor$ elements in $\overrightarrow{v}^o_i$
7 **else**
8 multiply by -1 the last $\lfloor m/2 \rfloor$ elements in $\overrightarrow{v}^o_i$
9 **if** $m \ mod \ 2 \ \neq \ 0$ **then**
10 randomly set to zero one of $\lceil m/2 \rceil$ unmodified element in $\overrightarrow{v}^o_i$
11 **end**

The factor $(\Sigma_{t+k} + \overline{\Sigma})$ in line 11 of Algorithm 3 is the *shifting operator* that allows to generate new individuals that are $\overline{\Sigma}$-shifted in the search space, while the factor $(V_{t+k} + \overline{V}^o)$ is the *orthogonal operator* which creates the new individuals that are rotated in the search space. Since the rotation is performed using orthogonal vectors, the new individuals will explore orthogonal regions not considered during the last k generations [15]. Fig. 1 shows the graphical interpretation of the two factors$(\Sigma_{t+k} + \overline{\Sigma})$ and $(V_{t+k} + \overline{V}^o)$ for two populations at generations t and $t+k$. We can see how $(\Sigma_{t+k} + \overline{\Sigma})$ allows to generate individuals which are shifted in the search space, while $(V_{t+k} + Orth(\overline{V}))$ creates new individuals with orthogonal evolution directions, i.e., new individuals which explore orthogonal and unexplored regions of the search space.

The remaining issue to solve is that the orthogonal individuals generated using SVD are not binary vectors, as the entries in P^o assume real values. To address this issue, in line 12 of Algorithm 3 we re-convert the entries in P^o into binary values by replacing elements lower than 0.5 with 0 and elements higher or equal to 0.5 with 1.

Finally, the new population P^* is obtained by replacing 50 percent of worst individuals in P_{t+k} with the orthogonal individuals in P^o, as shown at line 13 of Algorithm 3.

It is important to point out that this SVD-based diversity-preserving mechanism can be applied with any kind and number of objective criteria, since it works on the sets of solutions without modifying any evolutionary genetic operator or fitness function, as fitness sharing GA does [14].

3.3 Putting It Together

Algorithm 5 reports the novel DIV-GA, the variant of NSGA-II where we have integrated mechanisms to promote diversity in the initial population and during the evolution process.

Algorithm 5. DIV-GA

```
   Input:
   A test suite of size N
   Population size M
   SVD interval k
 1 begin
 2   t ⟵ 0
 3   P_t ⟵ ORTHOGONAL-POPULATION(N,M)
 4   old ⟵ t
 5   while not (stop condition) do
 6     //main loop of NSGA-II
 7     Q_t ⟵ MAKE-NEW-POP(P_t)
 8     R_t ⟵ P_t ⋃ Q_t
 9     𝔽 ⟵ FAST-NONDOMINATED-SORT(R_t)
10     P_{t+1} ⟵ ∅
11     d ⟵ 1
12     while |P_{t+1}| + |𝔽_d| ⩽ M do
13       CROWDING-DISTANCE-ASSIGNMENT(𝔽_d)
14       P_{t+1} ⟵ P_{t+1} ⋃ 𝔽_d
15       i ⟵ i + 1
16     Sort(𝔽_d) //according to the crowding distance
17     P_{t+1} ⟵ P_{t+1} ⋃ 𝔽_d[1 : (M − |P_{t+1}|)]
18     t ⟵ t + 1
19     if t mod k = 0 then
20       P_t ⟵ INJECT-DIVERSITY(P_old, P_t)
21       old ⟵ t
22   S ⟵ P_t
23 end
```

First and foremost, in line 3 a uniformly distributed population P_0 is created through the orthogonal design method, as described in Section 3.1. Then, the main loop of the DIV-GA algorithm first includes k executions of the NSGA-II algorithm, i.e., the loop between lines 6-18. During these k generations the usual selection, recombination, and mutation operators are used to create offsprings and the new population is created by selecting the best solutions between parents and offsprings. Then, every k generations we apply our SVD-based preserving technique on the past and current populations P_{old} and P_t respectively (lines 19-21).

Finally, the DIV-GA algorithm takes as an input the parameter k, which represents the temporal distance (in terms of number of iterations of the GA) between the two populations on which SVD has to be computed. As shown by De Lucia et al. [15], the injection of orthogonal individuals through SVD drastically reduces the number of iterations and the total convergence time of a GA, despite the cost of computing SVD. In other words, the lower the value of k, the higher the convergence speed of the GA. Therefore, in principle, SVD could be applied at each iteration ($k = 1$). However, in case the best individuals of two subsequent populations do not change, the SVD might have no effect and then the cost of computing SVD is not compensated by the benefits of injecting orthogonal individuals. In other words, the higher the value of k, the higher the probability that the best individuals of the two populations differ and then the higher the probability that SVD has effect on escaping from local optima. A systematic study on identifying the effects of k on the performances of a GA has not been done yet and is out of the scope of this paper. However, previous studies indicate that $k = 2$ provides generally good results both in real-coded numerical problems [15] and for evolutionary test data generation [38]. Thus, in this paper we also set $k = 2$.

4 Empirical Evaluation

The *goal* of this study is to evaluate DIV-GA, with the *purpose* of solving the test case selection problem. The *quality focus* of the study is represented in terms of three—possibly conflicting—objectives which are pursued when performing test case selection, namely increased code coverage capability, decreased execution cost, and increased past fault coverage.

The *context* of the study consists of 11 open-source and industrial programs available from the software-artifact infrastructure repository (SIR) [36]: `space`, an interpreter for Array Description Language, developed by European Space Agency; six GNU open-source programs `bash`, `flex`, `grep`, `gzip`, `sed` and `vim`; four programs of the Siemens suite, namely `printtokens`, `printtokens2`, `schedule`, and `schedule2`.

Table 1 reports the characteristics of the eleven programs, and specifically their size (in terms of LOC) and the size of the available test suites (in terms of number of test cases). As it can be noticed, the size of the selected programs ranges between 374 and 122,169 LOC, while the number of test cases between 215 and 13,583. The selection of these programs was not random. They have been used in previous work on regression testing and especially when experimenting techniques for the selection and prioritization of test suites [9], [14], [37], [56], [57], [67], [71], [72], [73], hence allowing us—wherever possible—to compare results.

In this study, we compare the performance of DIV-GA with two alternative test case selection techniques: (i) one based on additional greedy algorithm used by Rothermel et al. [53], and (ii) one based on the island version of NSGA-II (vNSGA-II) used by Yoo and Harman [71], [72].

4.1 Research Questions

The study aims to provide empirical evidence to answer the following research questions:

- **RQ_1**: *To What Extent Does DIV-GA Produce Near Optimal Solutions, Compared to Alternative Test Case Selection Techniques?* This research question aims at

TABLE 1
Programs Used in the Study

Program	LOC	# of Test Cases	Description
bash	59,846	1,200	Shell language interpreter
flex	10,459	567	Fast lexical analyser
grep	10,068	808	Regular expression utility
gzip	5,680	215	Data compression program
printtokens	726	4,130	Lexical analyzer
printtokens2	520	4,115	Lexical analyzer
schedule	412	2,650	Priority scheduler
schedule2	374	2,710	Priority scheduler
sed	14,427	360	Non-interactive text editor
space	6,199	13,583	European Space Agency program
vim	122,169	975	Improved *vi* editor

evaluating to what extent the proposed DIV-GA algorithm is able to produce a larger number of near optimal solutions, if compared to alternative, state-of-the-art test case selection techniques, namely additional greedy and vNSGA-II. Note that, due to the NP-complete nature of the test case selection problem, there is no solver able to find the exact optimal solutions with efficient computational cost, and all the experimented algorithms can provide only near-optimal solutions.

- **RQ_2**: *Which Is the Cost-Effectiveness of DIV-GA, Compared to Alternative Test Case Selection Techniques?* A common issue of test case selection techniques is that the reduced test suites might reveal less faults with respect to the original test suite because they contain a lower number of test cases to be executed against the software under test. With this second research question we are interested in understanding how many faults can be detected using the sub-test suites obtained by DIV-GA, in comparison with alternative techniques. This reflects the software engineer's needs to reduce a test suite without compromising the ability to detect source code faults.

The following sections describe all the experimented algorithms and the evaluation mechanisms performed to answer the aforementioned research questions.

4.2 Test Selection Objectives

In this paper we consider three different test case selection criteria, used in previous work [6], [72], [73], namely: (i) statement coverage, (ii) execution cost of test cases, and (iii) past fault coverage.

Statement coverage criterion. We measure statement coverage using the `gcov` tool part of the GNU C compiler (`gcc`). Specifically, we measure statement coverage, as also done by Yoo et al. [71], [72], [73]:

$$cov(X) = \frac{1}{m}\sum_{i=1}^{m} \phi_i, \tag{3}$$

where m is the total number of code statements to be covered and ϕ_i is equal to 1 if the ith statement is covered by at least one selected test case in X, 0 otherwise.

Execution cost criterion. As for the execution cost, in principle we could just measure the test case execution time. However, performing such a measure is not an easy task, because the measure depends on several external factors such as different hardware, application software, operating system, etc. In this paper we address this issue by counting the number of executed elementary instructions in the code, instead of measuring the actual execution time. This is consistent with what was done in previous work on multi-objective test case selection [71], [72]. We use the `gcov` tool to measure the execution frequency of every basic block (sequence of statements) composing each source code line. According to the tool documentation, a basic block is a linear section of code which has no branches and with only one entry point and only one exit. We considered the execution frequency of a basic block rather than the execution frequency of source code lines since a single source code line might contain multiple branches and calls (e.g., the `for` statement counts as three basic blocks, i.e., the initialization, the condition, and the increment). Therefore, the computational cost of each test case is approximated by summing the execution frequencies of all the executed basic blocks. Starting from these execution frequencies, the *execution cost criterion* is defined as follows:

$$cost(X) = \sum_{i=1}^{n} x_i \cdot cost(\tau_i), \tag{4}$$

where $cost(\tau_i)$ represents the execution frequency of the ith test case.

Past fault coverage criterion. As for the past fault coverage criterion, we consider the versions of the programs with seeded faults available in the SIR repository [36]. SIR also specifies whether or not each test case is able to reveal these faults. Such information can be used to assign a past fault coverage value to each test case subset, computed as the number of known past faults that this subset is able to reveal in the previous version. Specifically, starting from this history information, the past fault coverage (*fault*) achieved by a solution X can be measured as follows:

$$fault(X) = \frac{1}{h}\sum_{i=1}^{h} \varphi_i, \tag{5}$$

where h represents the number of test cases, while φ_i is equal to 1 if the ith past fault is covered by at least one selected test case in X, 0 otherwise.

Using the three test case selection criteria described above, we examine two and three-objectives formulations of

the test case selection problem. In particular, we consider the two-objective problem taking into account code coverage and execution cost as contrasting goals, similarly to what done in previous work [14], [71], [72]. Formally, the two-objective test case selection problem can be defined as follows:

Problem 1. Two-objective Test Case Selection Problem: *finding a set of optimal solutions X which maximizes the code coverage and minimizes the execution cost:*

$$\max cov(X) = \frac{1}{m}\sum_{i=1}^{m}\phi_{i,}$$
$$\min cost(X) = \sum_{i=1}^{n} x_i \cdot cost(\tau_i).$$

For the three-objective formulation, we add past fault detection history as a further objective, as also done in previous work [6], [71], [72]. The three-objective test case selection problem can be defined as follows:

Problem 2. Three-objective Test Case Selection Problem: *finding a set of optimal solutions X which maximizes the code coverage, minimizes the execution cost, and maximizes the past fault coverage:*

$$\max cov(X) = \frac{1}{m}\sum_{i=1}^{m}\phi_i,$$
$$\min cost(X) = \sum_{i=1}^{n} x_i \cdot cost(\tau_i),$$
$$\max fault(X) = \frac{1}{h}\sum_{i=1}^{h}\varphi_i.$$

Note that, besides the test case selection criteria defined above, it is possible to formulate other criteria, e.g., based on data-flow coverage or even functional requirements just providing a clear mapping between tests and criterion-based requirements. However, it is important to highlight that the goal of this work is not to analyze which set of test criteria is the most effective for regression testing. The formulations are used to illustrate how the diversity-preserving technique proposed in this paper can be applied to any number and kind of testing criteria to be satisfied.

4.3 Evaluated Algorithms

For the two-objective formulation of the test case selection problem, we compare the following optimization algorithms:

- The two-objective DIV-GA proposed in this paper (see Section 3) which uses SVD and orthogonal design to promote diversity between the selected test cases.
- The two-objective *additional greedy* algorithm used by Yoo and Harman [71] and by Rothermel et al. [53], which considers at same time both coverage and cost by maximizing the coverage per unit of time of the selected test cases (cost cognizant additional greedy), as reported in Algorithm 6. Specifically, such an algorithm starts with an empty set of test cases (line 3 of Algorithm 6) and iteratively picks the test case having the best additional coverage per unit cost (lines 5-6 of Algorithm 6). The process ends when the maximum code coverage is reached.
- The vNSGA-II algorithm, used by Yoo and Harman [71] for finding a set of non-dominated solutions that maximizes coverage while minimizing the cost of the selected test cases.

Algorithm 6. Cost Cognizant (or Two-Objective) Additional Greedy.

```
   Data:
   A test suite T = {t_1, ..., t_n}
   A set of program elements P covered by T
   Result: A set of sub-test suites S.
1  begin
2    C ⟵ ∅ // covered elements
3    S ⟵ ∅ // selected test cases
4    while C ⊂ P do
5      for each t_i ∈ T do
6        f_i = |cov(t_i) − C| / cost(t_i);
7      t_j ⟵ test case in T with minimum f_j
8      S ⟵ S ⋃ {t_j} // add t_j to solution
9      C ⟵ C ⋃ cov(t_j) // add cov(t_j) to covered elements
10     T ⟵ T − {t_j}
11     Add S to the Pareto set
12 end
```

Similarly, for what concerns the three-objective formulation of the test case selection problem, we compare the following optimization algorithms:

- The three-objective DIV-GA proposed in this paper (see Section 3).
- The three-objective *additional greedy* algorithm used by Yoo and Harman [71], which conflates code coverage, execution cost and past coverage in one objective function to be minimized, as highlighted in Algorithm 7. Such an algorithm is similar to Algorithm 6 with the only difference that at each iteration it picks the test case having the best additional code and past faults coverage per unit cost (lines 5-6 of Algorithm 7).
- The vNSGA-II algorithm used by Yoo and Harman [71] for finding a set of non-dominated solutions that maximizes code coverage and past fault coverage, while minimizing the cost of the selected test cases.

It is important to highlight that vNSGA-II—that we used as baseline for assessing the performance of DIV-GA—includes three different diversity-preserving mechanisms for promoting phenotype diversity between candidate solutions (set of selected test cases): *distance crowding*, *ranking based selection* and *migration between subpopulations* (islands). The proposed DIV-GA is based on the standard NSGA-II, so it also includes distance crowding and ranking based selection. However, DIV-GA also integrates the two novel diversity-preserving techniques introduced in this paper, which promotes diversity in the *genotype* space.

Algorithm 7. Three-Objectives Additional Greedy.

Data:
A test suite $T = \{t_1, \ldots, t_n\}$
A set of program elements P covered by T
Result: A set of sub-test suites S.

```
1  begin
2    C ⟵ ∅ // covered elements
3    F ⟵ ∅ // covered past faults
4    S ⟵ ∅ // selected test cases
5    while C ⊂ P do
6      for each t_i ∈ T do
7        f_i = (0.5 × |cov(t_i) − C| + 0.5 × |fault(t_i) − F|) / cost(t_i);
8      t_j ⟵ test case in T with minimum f_j
9      S ⟵ S ⋃ {t_j} // add t_j to solution
10     C ⟵ C ⋃ cov(t_j) // add cov(t_j) to covered elements
11     F ⟵ F ⋃ fault(t_j) // add fault(t_j) to covered faults
12     T ⟵ T − {t_j}
13     Add S to the Pareto set
14 end
```

All the algorithms have been implemented using MATLAB *Global Optimization Toolbox* [47] (release R2011b). In particular, we use the *gamultiobj* routine which implements both the standard version of the NSGA-II algorithm and its island version (vNSGA-II). The DIV-GA algorithm is also implemented by customizing the *gamultiobj* routine, while the generation of the initial population of DIV-GA based on orthogonal design is performed using the *rowexch* routine, which implements the *row-exchange algorithm* to generate m orthogonal arrays (m is the size of the initial population) of n factors (test cases) with 2-level ($\{0, 1\}$). For both vNSGA-II and DIV-GA we use the same parameters typically used for numerical problems [16]. Specifically:

- *Population size*. Since the search space of the test case selection problem is larger for programs with a larger test suite, we use different population sizes according to the size of the test suites to be optimized, as shown in Table 2.
- *Initial population*. For vNSGA-II the initial population is randomly generated within the solution space. For DIV-GA, the initial population is composed of the orthogonal arrays as explained in Section 3.1.
- *Number of generations*. The maximum number of generations varies according to the size of the test suites to be optimized. The values are reported in Table 2.
- *Crossover function*. We use a multi-point crossover, called *scattered crossover* with probability $p_c = 0.50$.
- *Mutation function*. We use a bit-flip mutation function with probability $p_m = 1/n$, where n is the size of the test suite (or equivalently n is the size of the chromosomes).
- *Stopping criterion*. The average change of the Pareto frontiers is lower than 5 percent for 50 subsequent generations, or the maximum number of generations is reached.
- *SVD frequency*. Orthogonal subpopulation are generated by SVD every two generations, i.e., $k = 2$. This parameter applies only for DIV-GA.

TABLE 2
Configurations of vNSGA-II and DIV-GA for the Programs Used in the Study

System	Population size	# of generations
bash	400	2,000
flex	400	1,000
grep	200	1,000
gzip	300	500
printtokens	200	500
printtokens2	200	500
schedule	200	500
schedule2	200	500
sed	300	1,000
space	400	1,000
vim	400	2,000

The setting of GA parameters has been performed using a MATLAB's routine, called *gaoptimset*, which allows to create a list of options to set all parameters. Both vNSGA-II and DIV-GA have been executed 30 times for each program under study, in order to account for their inherent randomness [3].

4.4 Evaluation Metrics

A simple way to evaluate the optimality of a multi-objective optimization algorithm consists of comparing its set of yielded solutions with those of the actual Pareto frontier. However, it is impossible to know *a priori* the actual Pareto frontier of the test case selection problem, because for large test suites it is unfeasible to compute the global optimal solution using an exhaustive search. In order to perform an *a posteriori* evaluation of the obtained Pareto frontiers, we construct a hybrid frontier by combining the best parts of the different frontiers achieved by all the algorithms (in all the runs), and considering only the solutions that are not dominated by the combined frontier. We call such a hybrid frontier *reference Pareto frontier* [71].

Formally, let $P = \{P_1, \ldots, P_l\}$ be the set of l different Pareto frontiers, the reference Pareto frontier P_{ref} is defined as follows:

$$P_{ref} \subseteq \bigcup_{i=1}^{l} P_i, \tag{6}$$

where P_{ref} is the maximal set of non-dominated solutions obtained by all the Pareto frontiers, i.e. $\forall p \in P_{ref} \ \nexists q \in P_{ref} : q \succ p$. Thus, P_{ref} helps to compare the global optimality of the different algorithms on the basis of the Pareto frontiers they produce.

We perform a comparison of the different algorithms by estimating two metrics widely used in global optimization problems:

- *Size of Pareto frontier*, that represents the number of non-dominated solutions obtained by each Pareto frontier P_i.
- *Number of non-dominated solutions*, that is the number of solutions that are not dominated by the reference Pareto frontier P_{ref}. Formally, it can be defined as the cardinality of the set $P_i^* = \{p \in P_i : \nexists q \in P_{ref} : q \succ p\}$. In other words, by definition of P_{ref}, P_i^* is the subset of P_i contained in the reference Pareto frontier P_{ref}.

These two metrics were used to answer RQ_1.

We also statistically analyze the obtained results, to check whether the differences between the solutions produced by two different algorithms are statistically significant or not. In particular, the values of the two employed metrics achieved by three algorithms over different independent runs were statistically compared using the *Welch's t test* [13] for both the multi-objective formulations (as done in previous work [14], [71]). Welch's t-test is generally used to test two groups with unequal variance, e.g., in our case the variance of the number of non-dominated solutions produces by the additional greedy and vNSGA-II or DIV-GA is different.[2] Significant p-values indicate that the corresponding null hypothesis can be rejected in favor of the alternative hypothesis, i.e., that one of the algorithms produced a Pareto frontier of larger size. In all our statistical tests we reject the null hypotheses for p-values < 0.05 (i.e., we accept a 5 percent chance of rejecting a null hypothesis when it is true [13]).

We preventively verify the applicability of the Welch's t-test on our data by performing the Wilk-Shapiro normality test. Such a test indicates a non significant deviation from normality (p-value > 0.05). Since we apply the Welch's t-test multiple times, we report both the original p-values and the corresponding adjusted ones obtained by using the Holm's correction procedure [35]. This procedure corrects the p-values resulting from n tests by sorting them in ascending order of values and multiplying the smallest by n, the next by $n-1$, and so on. We decided to report both original and adjusted p-values since the application of correction procedures is currently debated in software engineering community when assessing the performances of randomized algorithms [2].

Other than testing the hypotheses, we also estimated the magnitude of the difference between performances achieved by two algorithms. To this aim, we used the Cohen d effect size [13]. For dependent samples (to be used in the context of paired analyses, as in our study) the Cohen d effect size is defined as the difference between the means (M_1 and M_2), divided by the standard deviation of the (paired) differences between samples (σ_D):

$$d = \frac{M_1 - M_2}{\sigma_D}. \tag{7}$$

The effect size is considered *small* for $0.2 \leq |d| < 0.5$, *medium* for $0.5 \leq |d| < 0.8$ and *large* for $|d| \geq 0.8$ [11].

To address (**RQ**$_2$), we analyze the capability of optimized test suites—which may be composed of a smaller number of test cases than the original one—to detect faults. Since each algorithm provides more than one solution—i.e., more than one (near) optimal compromise between cost and coverage—to the best of our knowledge, there is no metric used in previous work to compare the effectiveness of two or more different sub-test suites. A simple way to perform such a comparison consists of plotting the percentage of faults detected by each solution provided by a given algorithm and the corresponding execution cost. This allows to graphically compare two or more Pareto frontiers, showing the percentage of detected faults at the same level of execution cost.

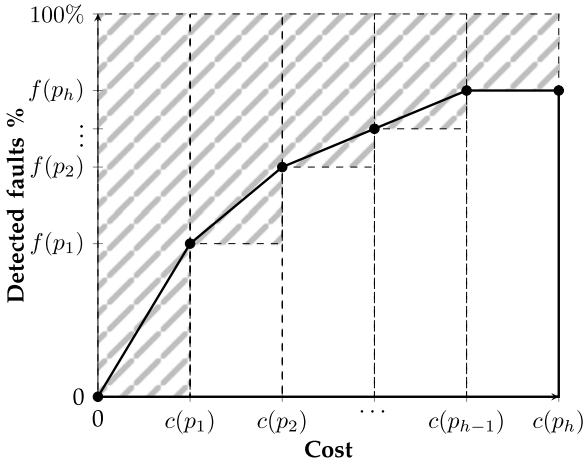


Fig. 2. Hypervolume metric based on *cost* and *effectiveness* of the sub-test suites.

In order to quantify the effectiveness of each Pareto frontier, we also use the hypervolume metric. Such a metric is generally used to measure the volume enclosed between a Pareto frontier $P = \{p_1, \ldots, p_h\}$ as compared to an ideal/optimal Pareto frontier R [4], [76]. In other words, the hypervolume measures the closeness of P to the ideal frontier R; the lower the hypervolume value, the better the optimality of the solutions in P [76]. Generally, the hypervolume can be computed within the space of the objectives to be optimized, or using external utility functions selected by the decision maker. In our case, we suggest revisiting the hypervolume metric by taking into account as external utility functions (i) the cost and (ii) the percentage of faults revealed by each solution (sub-set of the test suite) in the Pareto frontier. Hence, the goal is to measure how a Pareto frontier P is optimal from cost and effectiveness point of view. In this context, the ideal frontier R is represented by an ideal set of solutions (sub-test suites) that are able to reveal all the faults for any level of execution cost. According to the proposed utility functions, the weighted hypervolume indicator becomes a bi-dimensional indicator as shown in Fig. 2.

Without loss of generality, let $P = \{p_1, \ldots, p_h\}$ be a set of solutions, i.e., subsets of test cases. Let $f(p_i)$ be the percentage of faults revealed by the solution $p_i \in P$ and let $cost(p_i)$ be the corresponding execution cost. Let $R = \{r_1, ..., r_h\}$ be the corresponding ideal set of solutions, i.e. subsets of test cases that are able to reveal all faults at varying execution cost ($cost(r_i) = cost(p_i)$ and $f(r_i) = 1$ for each r_i). The hypervolume enclosed by these points can be easily computed as the sum of rectangles of width $[cost(p_{i+1}) - cost(p_i)]$ and height $[f(r_i) - f(p_i)]$, and then it is equal to:

$$I_H(P) = cost(p_1) + \sum_{i=1}^{h} [cost(p_{i+1}) - cost(p_i)] \cdot [1 - f(p_i)]. \tag{8}$$

Fig. 2 provides a graphical interpretation for the cost-effectiveness hypervolume of a given Pareto frontier. The hypervolume metric is an inverse function: the lower the

2. Since the additional greedy is a deterministic algorithm, the variance over 30 independent runs is zero. Conversely, because of the random inheritance of GAs, both vNSGA-II and DIV-GA do not reach a zero variance.

value, the better the average effectiveness of all the sub-test suites stated in the Pareto frontier.[3] Starting from the cost-effectiveness hypervolume metric, we can express it as percentage of the area (hypervolume) under the ideal/optimal frontier R as follows:

$$I_{CE}(P) = \frac{I_H(P)}{cost(p_h)}, \quad (9)$$

where $cost(p_h)$ denotes the execution cost of the last point in the Pareto frontier P. Such a metric measures the (cost-cognizant) weighted average percentage of faults detection loss of a Pareto frontier, i.e., the identified (near) optimal subsets of test suite. Note that we choose to not consider the reference Pareto frontier as the ideal frontier when computing $I_{CE}(P)$ because it is built by considering the best (non-dominated) solutions obtained by all the algorithms over all the independent runs. However, the solutions (sub-test suites) stated in the reference Pareto frontier might not reveal any fault or a sub-set of all regression faults. Hence, it cannot be considered as an ideal set of solutions from an effectiveness point of view, meant as its ability to detect faults.

5 Empirical Results

This section discusses the results of our study with the aim of answering the research questions formulated in Section 4.1.

5.1 RQ_1: To What Extent Does DIV-GA Produce Near Optimal Solutions, Compared to Alternative Test Case Selection Techniques?

Table 3 reports the size of Pareto frontiers and the number of non-dominated solutions for the two-objective test case selection problem obtained by (i) DIV-GA, (ii) the additional greedy algorithm, and (iii) vNSGA-II. Specifically, the table reports mean size and standard deviation over 30 independent runs of the algorithms. For all the 11 programs, the number of solutions forming the Pareto frontiers of DIV-GA is larger than the number of optimal solutions produced by the vNSGA-II and by the additional greedy algorithm. For example, on `gzip` DIV-GA provides a Pareto frontier with 222 solutions on average, while vNSGA-II provides 186 solutions, and the additional greedy algorithm only 19. This represents a substantial improvement from the tester's perspective, since a larger Pareto frontier provides a wider range of candidate solutions that provide different compromises between cost and coverage.

The analysis of Table 3 reveals that there is no clear winner among the additional greedy and vNSGA-II, confirming the results of previous study [71]. In five cases out of 11, the size of the Pareto frontiers obtained by vNSGA-II is smaller than the Pareto frontiers obtained by the additional greedy, while for the remaining cases the size for vNSGA-II is larger.

DIV-GA also turned out to be able to produce a larger number of non-dominated sub-test suites with respect to all the other algorithms. In particular, the majority of subset of tests forming the Pareto frontiers of DIV-GA are also non-dominated by any other algorithm. For example, on `bash` the Pareto frontier produced by DIV-GA contains 354 solutions, and among them 311 solutions are non-dominated by those obtained by all the other algorithms (i.e., such solutions are stated in the reference Pareto frontiers), while the number of non-dominated solutions provided by the additional greedy algorithm and vNSGA-II is really small, especially when compared with the total amount of provided solutions. For example, on `bash` the additional greedy provides 232 different solutions. However, among them only 30 solutions (less than 13 percent) are non-dominated by the solutions of the other algorithms (in particular by those of DIV-GA). Similarly, vNSGA-II obtains 276 different solutions, but only 13 percent of them are non-dominated by those of the other algorithms. In summary, for the two-objective formulation, DIV-GA not only produces more Pareto optimal sub-test suites than the other algorithms, however these sub-test suites provide better code coverage

TABLE 3
Two-Objective Test Case Selection: Mean Pareto Sizes and Number of Non-Dominated Solutions Achieved by the Different Algorithms

Program	Method	Pareto size		Non-Dominated Solutions	
		Mean	St. Dev.	Mean	St. Dev.
bash	DIV-GA	**354**	4.98	**311**	35.39
	Add. Greedy	232	-	30.20	27.24
	vNSGA-II	276	5.69	37	52.04
flex	DIV-GA	**306**	8.52	**303**	8.23
	Add. Greedy	43	-	7	0
	vNSGA-II	157	70.99	0	-
grep	DIV-GA	**162**	2.19	**140**	3.05
	Add. Greedy	70	-	9	-
	vNSGA-II	60	-	3.75	4.79
gzip	DIV-GA	**222**	3.71	**186**	13.29
	Add. Greedy	19	-	5.67	-
	vNSGA-II	88	1.36	30	13.48
sed	DIV-GA	**270**	-	**252**	18.03
	Add. Greedy	33	-	3	-
	vNSGA-II	225	33.72	27.45	39.71
printtokens	DIV-GA	**86**	5.26	**55.64**	11.75
	Add. Greedy	10	-	3	-
	vNSGA-II	6.60	2.30	0	-
printtokens2	DIV-GA	**96**	11.31	**63**	18.57
	Add. Greedy	10	-	4	-
	vNSGA-II	23.60	7.50	0	-
schedule	DIV-GA	**62.22**	3.03	**28.33**	2.69
	Add. Greedy	6	-	2	-
	vNSGA-II	1	-	0	-
schedule2	DIV-GA	**63.22**	5.49	**31.14**	4.45
	Add. Greedy	9	-	4	-
	vNSGA-II	18	0.58	0	-
space	DIV-GA	**344**	4.19	**340**	12.39
	Add. Greedy	119	-	3	-
	vNSGA-II	284	85.58	6.67	14.38
vim	DIV-GA	**353**	1.10	**234**	75.37
	Add. Greedy	266	-	91	77.78
	vNSGA-II	339	8.66	6.5	9.19

The best result for each program is highlighted in bold face.

3. The hypervolume measures the closeness of the Pareto frontier P to the ideal effectiveness: all the solutions stated in P are able to reveal all the faults.

TABLE 4
Comparison between Different Algorithms for the Two-Objective Test Case Selection Problem

Program	Hypothesis			Pareto Size			Non Dom. Solutions		
				p-values	Adjusted p-values	Cohen's d	p-values	Adjusted p-values	Cohen's d
bash	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	58.39 (L)	<**0.01**	<**0.01**	8.90 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	16.64 (L)	<**0.01**	<**0.01**	6.16 (L)
	Add. Greedy	>	vNSGA-II	1	1	−10.96 (L)	0.58	0.58	0.16
flex	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	40.92 (L)	<**0.01**	<**0.01**	50.96 (L)
	DIV-GA	>	vNSGA-II	0.37	0.75	2.94 (L)	<**0.01**	<**0.01**	52.69 (L)
	Add. Greedy	>	vNSGA-II	1	1	−2.04 (L)	<**0.01**	<**0.01**	-
grep	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	59.13 (L)	<**0.01**	<**0.01**	56.11 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	12.37 (L)	<**0.01**	<**0.01**	34.05 (L)
	Add. Greedy	>	vNSGA-II	<**0.01**	<**0.01**	2.30 (L)	<**0.01**	<**0.01**	1.44 (L)
gzip	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	77.31 (L)	<**0.01**	<**0.01**	19.19 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	47.92 (L)	<**0.01**	<**0.01**	11.67 (L)
	Add. Greedy	>	vNSGA-II	1	1	−71.41 (L)	1	1	−2.56 (L)
printtokens	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	19.47 (L)	<**0.01**	<**0.01**	6.39 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	18.59 (L)	<**0.01**	<**0.01**	6.65 (L)
	Add. Greedy	>	vNSGA-II	<**0.01**	<**0.01**	1.68 (L)	<**0.01**	<**0.01**	4.32 (L)
printtokens2	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	10.71 (L)	<**0.01**	<**0.01**	2.86 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	6.39 (L)	<**0.01**	<**0.01**	3.04 (L)
	Add. Greedy	>	vNSGA-II	0.41	0.41	0.14 (L)	<**0.01**	<**0.01**	14.32 (L)
schedule	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	26.22 (L)	<**0.01**	<**0.01**	13.83 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	28.33 (L)	<**0.01**	<**0.01**	14.19 (L)
	Add. Greedy	>	vNSGA-II	0.78	0.78	20.74 (L)	**0.02**	**0.02**	3.77 (L)
schedule2	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	16.95 (L)	<**0.01**	<**0.01**	8.44 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	11.49 (L)	<**0.01**	<**0.01**	9.66 (L)
	Add. Greedy	>	vNSGA-II	1	1	−25.56 (L)	**0.03**	**0.03**	4.54 (L)
sed	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	70.57 (L)	<**0.01**	<**0.01**	19.52 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	3.10 (L)	0.14	0.28	7.28 (L)
	Add. Greedy	>	vNSGA-II	1	1	−11.06 (L)	0.97	0.97	−0.88 (L)
space	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	76.12 (L)	<**0.01**	<**0.01**	38.41 (L)
	DIV-GA	>	vNSGA-II	**0.02**	**0.05**	1.00 (L)	<**0.01**	<**0.01**	26.22 (L)
	Add. Greedy	>	vNSGA-II	1	1	2.72 (L)	0.69	0.69	−0.22 (S)
vim	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	12.57 (L)	<**0.01**	<**0.01**	1.86 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	4.18 (L)	<**0.01**	<**0.01**	4.17 (L)
	Add. Greedy	>	vNSGA-II	1	1	17.50(L)	<**0.01**	<**0.01**	1.45 (L)

Welch's t-test p-values, adjusted p-values, and Cohen's d effect size. We use S, M, and L to indicate small, medium and large effect sizes, respectively. p-values that are statistically significant (i.e., p-value < 0.05) are reported in bold face.

with lower execution cost than the solutions produced by the other algorithms.

The considerations above are also supported by statistical analyses. Table 4 reports the results of the Welch's t-test and Cohen's d effect size, obtained comparing (across the 30 GA runs) the size of the Pareto frontiers and the number of non-dominated solutions achieved by the experimented algorithms.[4] The statistical analysis confirms that DIV-GA always produces Pareto frontiers that are significantly larger than those produced by the additional greedy algorithm (in 100 percent of cases) and by vNSGA-II (in 82 percent of cases). The corresponding effect size values reveal that the magnitude of the improvement of DIV-GA over the other two algorithms is large ($d > 1$) in the majority of cases: 100 percent of the cases for the additional greedy and 91 percent of the cases for vNSGA-II.

4. As explained in Section 4.4, besides showing the p-values we also show the p-value adjusted using the Holm's correction procedure [35].

As for the number of solutions DIV-GA statistically outperforms the other two algorithms with a large effect size in all the cases. When comparing the additional greedy and vNSGA-II, we can also note that for all the 11 programs none of the two algorithms turns out to be statistically better than the other in terms of Pareto optimality. In fact, in some cases the additional greedy significantly outperforms vNSGA-II, while other cases the opposite happens.

Fig. 3 provides—for some programs considered in our study, i.e., `flex`, `grep`, `gzip` and `printtokens`—a graphical comparison between the Pareto frontiers obtained by the three algorithms and a "reference" Pareto frontier, built as explained in Section 4.4. We obtained consistent results for all other programs as well (see [51] for further details).

As it can be noticed, the Pareto frontiers provided by DIV-GA are much closer to the reference Pareto frontiers (often the two frontiers are perfectly overlapped) than the Pareto frontiers provided by vNSGA-II and the additional greedy. The additional greedy algorithm provides solutions that are, in some cases, quite close to the reference frontiers.

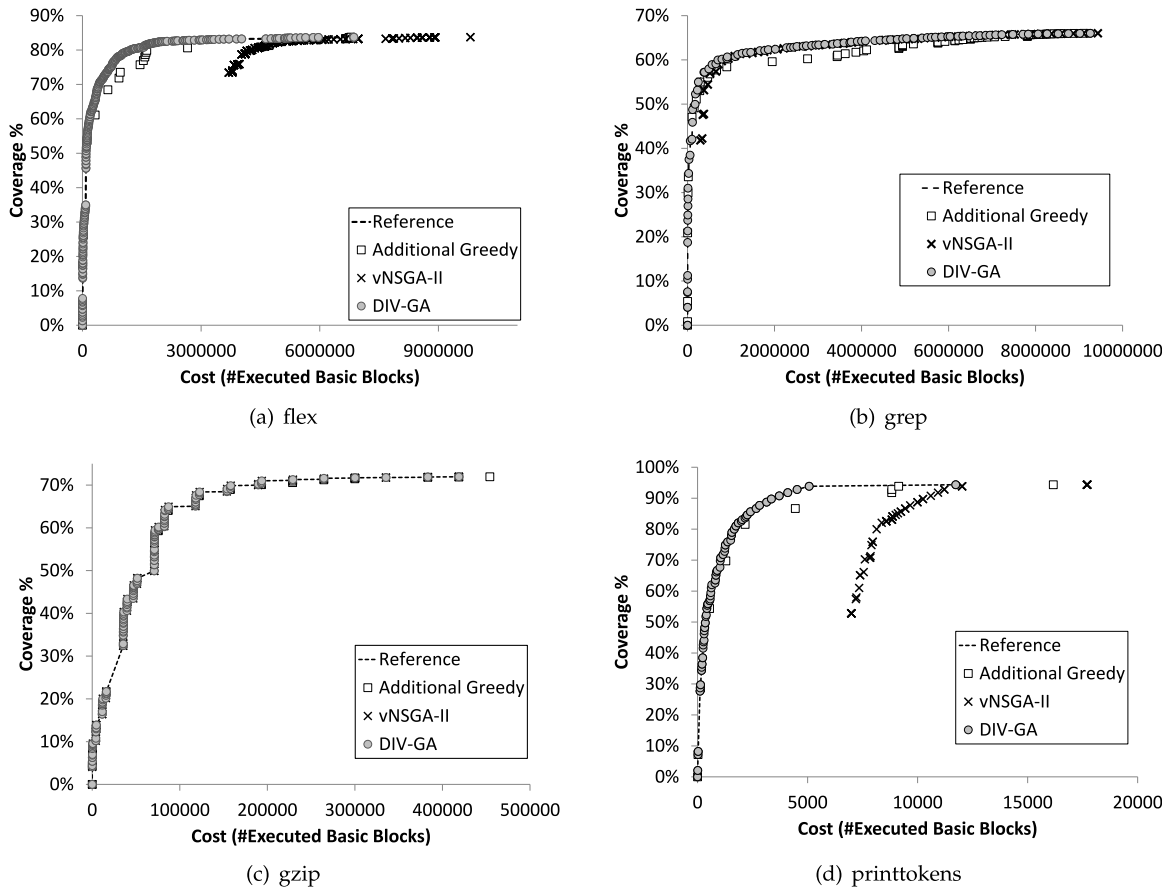


Fig. 3. Pareto frontiers.

However, the majority of them are dominated by solutions produced by DIV-GA. Instead, vNSGA-II produces solutions quite far from the optimal set of solutions (e.g., on `printtokens`). Only on `gzip`, vNSGA-II provides Pareto fronts that are close to the reference frontiers. However, such algorithms provided a lower number of solutions with respect to DIV-GA (this explains the substantial difference in terms of Pareto frontier size reported in Table 3) and also such solutions are slightly worse than the reference frontier (this explains the lower number of non-dominated solutions reported in Table 3). Particularly interesting are the results obtained for all the programs in the Siemens suite—i.e., `printtokens`, `printtokens2`, `schedule`, and `schedule2`—where vNSGA-II is very far from the optimal Pareto frontier while DIV-GA provides (near) optimal frontiers. Finally, DIV-GA provides a larger number of non-dominated solutions that also cover a wider range of coverage and execution cost values with respect to vNSGA-II. In particular, vNSGA-II is not able to generate solutions (sub-sets of the test suite) with low-coverage and low-execution cost (code coverage is always greater than 50 percent). Instead, DIV-GA provides solutions with code coverage values ranging from 0 percent to the maximum coverage value.

Table 5 compares the performance of the three experimented algorithms for the three-objective formulation of the test case selection problem. Also in this case, results are shown in terms of mean and standard deviation of the size of Pareto frontiers and number of non-dominated solutions computed across 30 independent runs of each algorithm.

In all cases, the size of the Pareto frontiers obtained by DIV-GA is larger than those obtained using the additional greedy algorithm. For example, on `gzip` DIV-GA provides 198 solutions against 19 solutions produced using the additional greedy. DIV-GA also produces larger Pareto frontiers than vNSGA-II in nine out of 11 cases, while on `space` and `gzip` the size of the Pareto frontiers is exactly the same. However, by looking at the non-dominance of the solutions, DIV-GA always produces a number of non-dominated solutions larger than the number of non-dominated solutions produced by the other two algorithms. Such solutions are also stated on the reference frontier, i.e., they are not dominated by solutions produced by the other algorithms. Conversely, the majority of solutions produced by the additional greedy and vNSGA-II are dominated by—i.e., they are worse than—the solutions of DIV-GA. For example, on `flex` the additional greedy algorithm produces 47 solutions forming the Pareto frontier, and among them only seven solutions (less than 15 percent of the total amount of the produced solutions) are non-dominated by any other algorithm. Similarly, vNSGA-II produces 139 solutions for the same program, but none of them belongs to the reference Pareto frontier, i.e., all of them are dominated by other solutions. Hence, for the three-objective test case selection problem, DIV-GA produces a larger number of sub-test suites with higher code coverage, higher past fault coverage and lower execution cost than both vNSGA-II and the additional greedy.

TABLE 5
Three-Objective Formulation of the Test Case Selection Problem: Mean Size of Pareto Frontier and Mean Number of Non-Dominated Solutions Obtained by the Different Algorithms

Program	Method	Pareto size		Non-Dominated Solutions	
		Mean	St. Dev.	Mean	St. Dev.
bash	DIV-GA	**354**	2.98	**310**	53.73
	Add. Greedy	233	-	31	-
	vNSGA-II	276	4.47	48	69.20
flex	DIV-GA	**331**	8.06	**328**	8.04
	Add. Greedy	47	-	7	-
	vNSGA-II	139	44.62	0	-
grep	DIV-GA	**317**	13.06	**294**	44.21
	Add. Greedy	72	-	6	-
	vNSGA-II	207	46.23	21	42.73
gzip	DIV-GA	**198**	4.97	**171**	11.30
	Add. Greedy	19	-	1	-
	vNSGA-II	**195**	4.66	130	20.77
printtokens	DIV-GA	**110**	33.28	**110**	33.71
	Add. Greedy	13	-	7	-
	vNSGA-II	6	2.77	0	-
printtokens2	DIV-GA	**190**	30.11	**189**	27.70
	Add. Greedy	11	-	4	-
	vNSGA-II	11	3.50	0	-
schedule	DIV-GA	**213**	11.33	**199**	11.29
	Add. Greedy	10	-	6	-
	vNSGA-II	123	21.48	18.67	30.24
schedule2	DIV-GA	**136**	23.09	**91**	27.77
	Add. Greedy	11	-	1	-
	vNSGA-II	118	20.47	9	20.20
sed	DIV-GA	**195**	38.02	**164**	43.48
	Add. Greedy	33	-	5	-
	vNSGA-II	98	19.83	36.14	12.56
space	DIV-GA	**360**	-	**318**	59.51
	Add. Greedy	126	-	7	-
	vNSGA-II	**360**	-	78	91.65
vim	DIV-GA	**393**	**1.77**	**219**	**16.29**
	Add. Greedy	266	-	163	-
	vNSGA-II	182	3.50	26.43	10.75

The best result for each program is highlighted in bold face.

Results of the Welch's t test confirm that the differences between DIV-GA and the other two algorithms are also statistically significant. Table 6 reports the p-values obtained comparing the Pareto frontier size and the number of non-dominated solutions achieved by the experimented algorithms (the p-values have been adjusted using the Holm's [35] correction procedure). For all programs, the Pareto frontiers produced by DIV-GA are significantly larger than those produced by the additional greedy (100 percent of cases) and by vNSGA-II (82 percent of cases) with a very large effect size in all the cases. When comparing vNSGA-II and the additional greedy, we can also note that vNSGA-II produces a larger number of sub-test suites than the additional greedy.

DIV-GA also always produces a larger number of solutions—i.e., more sub-test suites—that are non-dominated by any solution obtained by the other algorithms. The results also show that in general the additional greedy algorithm is dominated by vNSGA-II. However, on some programs—i.e. `flex`, `grep`, `printtokens`, `printokens2`, and `vim`—the additional greedy algorithm produces more optimal results than vNSGA-II, as was also pointed out by Yoo and Harman [71].

Figs. 4, 5, and 6 show the results for the three-objective formulation on three programs, i.e., `flex`, `gzip`, and `printtokens`. Consistent results have been obtained for all the other programs (see the on-line Appendix [51] for further details). The 3D plots displays the solutions produced by (i) DIV-GA, (ii) the additional greedy algorithm, (iii) vNSGA-II, and (iv) the reference Pareto frontier (denoted using black dots). The additional greedy algorithm produces solutions that are quite close to the reference frontier. However the number of the produced solutions is really small if compared to the reference frontier. DIV-GA always produces three-objective solutions stated in the reference frontier: in all cases more than 90 percent of the solutions obtained by DIV-GA overlap the reference frontier. Hence, DIV-GA always produces solutions that are non-dominated by any other algorithm. Instead, vNSGA-II produces (near) optimal solutions only in a few cases. For example, on `gzip` the Pareto frontier obtained by vNSGA-II is quite close to the reference Pareto frontier, even if Table 6 reveals that only a part of such solutions are non-dominated, while on `printtokens`, the solutions obtained by vNSGA-II are quite far from the reference Pareto frontier.

RQ_1 summary. For both the two- and three-objective test case selection problems, we can conclude that DIV-GA is always able to produce more Pareto-optimal sub-test suites than the additional greedy algorithm and vNSGA-II. Such sub-test suites represent Pareto-optimal compromises between coverage and cost.

5.2 RQ_2: Which Is the Cost-Effectiveness of DIV-GA, Compared to Alternative Test Case Selection Techniques?

Table 7 reports the mean values of the cost-effectiveness hypervolume metric (I_{EC}) related to the different Pareto frontiers produced by the three experimented algorithms: (i) DIV-GA, (ii) additional greedy, and (iii) vNSGA-II. The reported values represent the mean of the I_{EC} values achieved over 30 independent runs. Results are also collected according to the two formulations of test case selection problem investigated in this paper.

For the two-objective formulation, in 10 out of 11 programs the hypervolume values obtained by DIV-GA are smaller than those achieved by the additional greedy. Hence, the test cases stated in the corresponding Pareto frontiers are able to detect more faults with a lower execution cost (which mirrors a lower average percentage of fault detection loss). Only on `sed`, the additional greedy produces the same hypervolume values as DIV-GA. A similar analysis can be done by comparing DIV-GA and vNSGA-II: for all programs, the I_{EC} values provided by DIV-GA are better then those provided by vNSGA-II. When comparing the additional greedy and vNSGA-II, it is possible to observe that there is no clear winner among them, also from an effectiveness point of view. In two out of 11 cases, the test cases selected by vNSGA-II can detect more faults than the solutions detected by the additional greedy algorithm, while in six out of 11

TABLE 6
Comparison between Different Algorithms for the Three-Objective Test Case Selection Problem

Program	Hypothesis			Pareto Size			Non Dom. Solutions		
				p-values	Adjusted p-values	Cohen's d	p-values	Adjusted p-values	Cohen's d
bash	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	57.48 (L)	<**0.01**	<**0.01**	7.31 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	20.59 (L)	<**0.01**	<**0.01**	4.22 (L)
	Add. Greedy	>	vNSGA-II	1	1	−13.60 (L)	0.73	0.73	0.34 (S)
flex	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	49.82 (L)	<**0.01**	<**0.01**	56.40 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	5.96 (L)	<**0.01**	<**0.01**	57.49 (L)
	Add. Greedy	>	vNSGA-II	1	1	−2.94 (L)	<**0.01**	<**0.01**	20.62
grep	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	26.56 (L)	<**0.01**	<**0.01**	9.21 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	3.25 (L)	<**0.01**	<**0.01**	6.28 (L)
	Add. Greedy	>	vNSGA-II	1	1	−4.12 (L)	<**0.01**	<**0.01**	0.49 (M)
gzip	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	50.88 (L)	<**0.01**	<**0.01**	56.00 (L)
	DIV-GA	>	vNSGA-II	0.12	0.25	0.62 (M)	<**0.01**	<**0.01**	4.46 (L)
	Add. Greedy	>	vNSGA-II	1	1	−53.37 (L)	1	1	−8.81 (L)
printtokens	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	4.14 (L)	<**0.01**	<**0.01**	4.39 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	4.43 (L)	<**0.01**	<**0.01**	4.66 (L)
	Add. Greedy	>	vNSGA-II	<**0.01**	<**0.01**	3.68 (L)	<**0.01**	<**0.01**	19.45 (L)
printtokens2	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	8.41 (L)	<**0.01**	<**0.01**	9.42 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	8.377 (L)	<**0.01**	<**0.01**	9.62 (L)
	Add. Greedy	>	vNSGA-II	0.40	0.40	0.12	<**0.01**	<**0.01**	12.13 (L)
schedule	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	9.12 (L)	<**0.01**	<**0.01**	24.21 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	2.63 (L)	<**0.01**	<**0.01**	7.91 (L)
	Add. Greedy	>	vNSGA-II	1	1	−7.46 (L)	0.82	0.82	0.59 (M)
schedule2	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	7.63 (L)	<**0.01**	<**0.01**	4.57 (L)
	DIV-GA	>	vNSGA-II	0.07	0.14	0.80 (L)	<**0.01**	<**0.01**	3.38 (L)
	Add. Greedy	>	vNSGA-II	1	1	−7.39 (L)	0.94	0.94	0.55 (M)
sed	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	6.03 (L)	<**0.01**	<**0.01**	7.07 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	3.19 (L)	<**0.01**	<**0.01**	5.62 (L)
	Add. Greedy	>	vNSGA-II	1	1	−4.65 (L)	1	1	3.51 (L)
space	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	251.46 (L)	<**0.01**	<**0.01**	8.98 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	145.73 (L)	<**0.01**	<**0.01**	1.02 (L)
	Add. Greedy	>	vNSGA-II	1	1	−83.09 (L)	0.93	0.93	−2.64 (L)
vim	DIV-GA	>	Add. Greedy	<**0.01**	<**0.01**	106.25 (L)	<**0.01**	<**0.01**	4.88 (L)
	DIV-GA	>	vNSGA-II	<**0.01**	<**0.01**	76.70 (L)	<**0.01**	<**0.01**	13.97 (L)
	Add. Greedy	>	vNSGA-II	<**0.01**	<**0.01**	33.84 (L)	<**0.01**	<**0.01**	17.96 (L)

Welch's t-test p-values, adjusted p-values, and Cohen's d effect size. We use S, M, and L to indicate small, medium and large effect sizes respectively. Statistically significant p-values (i.e., < 0.05) are reported in bold face.

cases the greedy algorithm outperforms vNSGA-II. In the remaining three cases, there is no difference between the I_{EC} values achieved by the two algorithms. This result indicates that *injecting diversity* in GAs improves the effectiveness of multi-objective GAs (vNSGA-II in our case).

To provide a graphical interpretation to the I_{EC} metric, Fig. 7 plots the percentage of faults detected by the solutions (sub-test suites) provided by the three algorithms at same level of execution cost on `space` and `printtokens`. We can observe that the sub-test suites selected by DIV-GA are able to detect more faults than

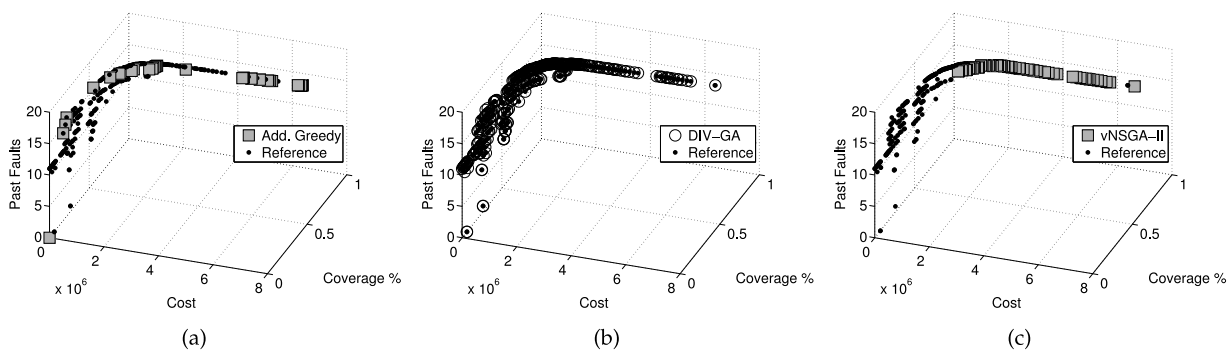


Fig. 4. Three-objective Pareto Frontiers achieved on *flex*.

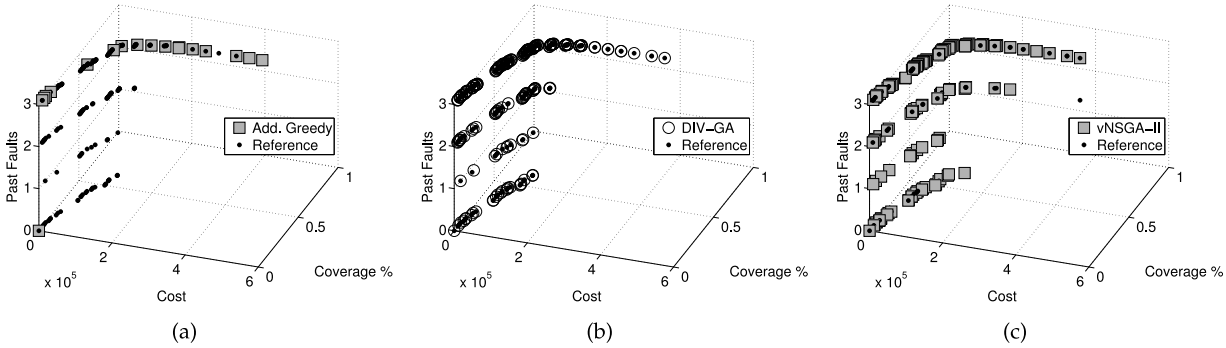


Fig. 5. Three-objective Pareto Frontiers achieved on *gzip*.

the additional greedy with lower execution cost. For example, on space the test suites optimized by DIV-GA can detect 100 percent of faults, while the percentage of faults produced by the other two algorithms is lower at the same level of execution cost. Similar considerations can be made on the other programs (see the on-line Appendix [51] for further details).

For the three-objective formulation of the test case selection problem, we obtained results that are quite similar to those obtained for the two-objective formulation (see Table 7). Indeed, for all the programs, the cost-effectiveness hypervolume values achieved by DIV-GA are smaller than those achieved by the additional greedy. This means that the test cases within the corresponding Pareto frontiers are able to detect more faults with a lower execution cost, mirroring a lower average percentage of fault detection loss per unit cost. A similar analysis can be done by comparing DIV-GA and vNSGA-II. In general, DIV-GA obtains the best hypervolume values. Only on grep and sed the I_{CE} values obtained by DIV-GA and vNSGA-II are the same. When comparing the additional greedy and vNSGA-II, we can observe that in general the additional greedy is better in terms of fault detection-effectiveness. In two out of 11 cases, the test cases selected by vNSGA-II can detect more faults than the test cases selected by the additional greedy algorithm, while in eight out of 11 cases the additional greedy outperforms vNSGA-II. Finally, in only one case there is no difference between the I_{EC} values produced by the two algorithms.

Fig. 8 plots the cost/faults curves obtained by the three algorithms. The goal of such an analysis is to provide a graphical comparison of the percentage of faults detected by the different solutions (sub-test suites) at same level of execution cost. We can notice that the sub-test suites obtained by DIV-GA are able to detect more solutions than both the additional greedy and vNSGA-II with a lower (or in some cases the same) execution cost. For example, on space, the test suites optimized by DIV-GA can detect 100 percent of the faults, while the percentage of faults detected by the other two algorithms is lower for the same level of execution cost. On printtokens, all the algorithms provide solution that are able to reveal all faults. However, DIV-GA turned out to be better than the other techniques in terms of execution cost. Similar considerations can be made on the other programs (see the on-line Appendix [51] for further details).

RQ_2 summary. For both two and three-objective formulations of the test case selection problem, we can conclude that DIV-GA is always able to produce optimal sub-test suites within the Pareto frontiers. Such sub-test suites are able to reveal more faults than the sub-test suites obtained by the additional greedy algorithm and vNSGA-II. Moreover, the corresponding execution cost is lower than the other techniques.

6 Additional Analyses

In this section we provide additional empirical analyses to assess the execution time of the experimented algorithms, the benefits of injecting diversity in the genotype space and the individual contribution of the orthogonal design defining the initial population and of the orthogonal evolution of the population through SVD. Although these analyses are not directly part of the study described in Sections 4 and 5, they help to understand the

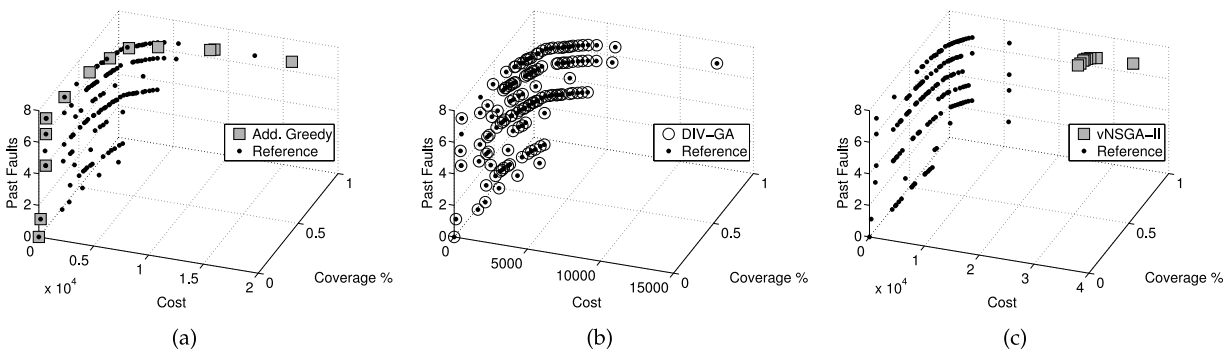


Fig. 6. Three-objective Pareto Frontiers achieved on *printtokens*.

TABLE 7
Mean Cost-Effectiveness Hypervolume Values

Program	Method	I_{CE}	
		2-Objective	3-Objective
bash	Add. Greedy	0.45	0.32
	DIV-GA	**0.39**	**0.30**
	vNSGA-II	0.52	0.48
flex	Add. Greedy	0.03	0.15
	DIV-GA	**0.02**	**0.05**
	vNSGA-II	0.04	0.20
grep	Add. Greedy	0.51	0.51
	DIV-GA	**0.40**	**0.27**
	vNSGA-II	**0.40**	0.28
gzip	Add. Greedy	0.56	0.52
	DIV-GA	**0.51**	**0.51**
	vNSGA-II	0.60	0.54
printtokens	Add. Greedy	1	0.72
	DIV-GA	**0.83**	**0.60**
	vNSGA-II	0.97	0.81
printtokens2	Add. Greedy	0.86	0.17
	DIV-GA	**0.81**	**0.08**
	vNSGA-II	1	0.86
schedule	Add. Greedy	1	0.07
	DIV-GA	**0.97**	**0.06**
	vNSGA-II	0.98	0.10
schedule2	Add. Greedy	1	0.89
	DIV-GA	**0.99**	**0.84**
	vNSGA-II	1	0.89
sed	Add. Greedy	**0.23**	0.20
	DIV-GA	**0.23**	**0.10**
	vNSGA-II	0.28	**0.10**
space	Add. Greedy	0.35	0.14
	DIV-GA	**0.24**	**0.12**
	vNSGA-II	0.35	0.16
vim	Add. Greedy	0.24	0.23
	DIV-GA	**0.22**	**0.19**
	vNSGA-II	0.33	0.25

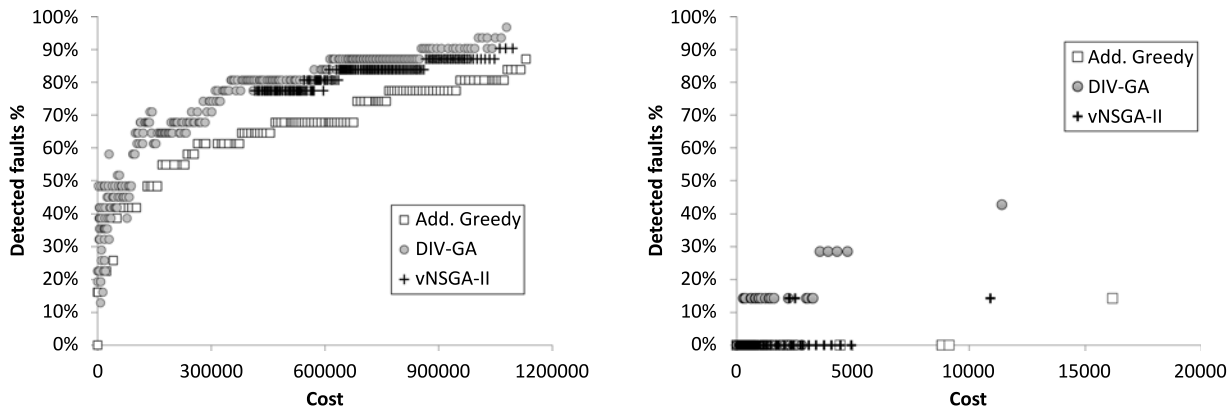
The best result for each program is highlighted in bold face.

performance of the proposed and experimented algorithm, as well as to understand the factors that could have played an important role in the performance of DIV-GA.

6.1 Execution Time of the Experimented Algorithms

The results show that DIV-GA provides better results than the other two algorithms with respect to the investigated research questions. However, DIV-GA introduces the computation of SVD—which is known to be quite expensive—in the main loop of a GA. Previous studies [15] have shown that injecting diversity through SVD in GAs drastically increases the convergence speed of the algorithm, thus more than compensating the extra time required to periodically compute SVD. In this section we compare the execution times of the different algorithms experimented in our empirical study. Table 8 reports the average execution time required by each algorithm for each software program used in the empirical study. The execution time was measured using a machine with Intel Core i7 processor running at 2.40 GHz with 8 GB RAM. For both two- and three-objective formulation, we can note that DIV-GA requires less execution time for its convergence with respect to vNSGA-II. Specifically, DIV-GA takes 52 percent of the execution time required by vNSGA-II for the same programs, on average. This is an important improvement if we also consider that DIV-GA not only is much faster than vNSGA-II, but it provides more optimal sub-test suites ($\mathbf{RQ}_1$) that are able to reveal more faults ($\mathbf{RQ}_2$).

When comparing the GAs with the addition greedy, as expected we found that the greedy algorithm is the fastest one in general. However, for the two largest systems (i.e., `bash` and `vim`) the greedy algorithm turned out to be the worst one, requiring more than 40 minutes for its execution against 20 minutes on average required by DIV-GA. This results is not surprising since, as explained by Harrold et al. [29] the greedy algorithm has an execution time of $O(|T| \cdot \max|T_i|)$, where $|T|$ represents the size of the original test suite, while $\max|T_i|$ denotes the cardinality of the largest group of test cases which is able to reach the maximum coverage. For large systems, $\max|T_i|$ might increase a lot because the number of test cases (and then the number of iterations for the additional greedy) required to reach the maximum coverage can be very high. For example, with the two-objective formulation the additional greedy selected 232 test cases on `bash`, which also corresponds to the number of its iterations; while on a small program such as `schedule` the additional greedy selected



(a) space (b) printtokens

Fig. 7. Effectiveness of the achieved sub-test suites for two-objective test case selection.

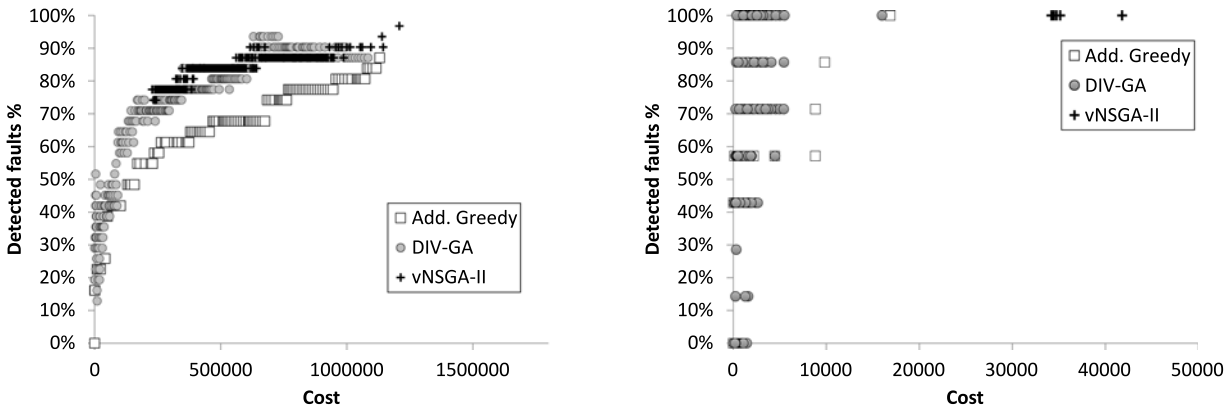


Fig. 8. Effectiveness of the achieved sub-test suites for three-objectives test case selection.

only nine test cases, i.e. it performed only nine iterations. These results motivate the investigation of new fast meta-heuristics against the additional greedy algorithms especially for larger programs.

6.2 Effect of Injecting Diversity in the Genotype Space

Most approaches dealing with the genetic drift (e.g., distance crowding) inject diversity in the phenotype space which also naturally mirrors increasing diversity in the genotype space. Conversely, the approach proposed in this paper injects diversity in the genotype space. Specifically, our conjecture is that injecting diversity in the genotype space also results in an increment of the diversity in the phenotype space. The results of our empirical study confirm our conjecture showing that DIV-GA generally overcomes previous approaches. However, to provide further empirical evidence that injecting diversity in the genotype space through orthogonal design and orthogonal evolution also has an effect on the phenotype diversity, we compare the average phenotype diversity of the solutions obtained by DIV-GA and vNSGA-II when varying the number of generations. At each generation a MOGA (i.e, DIV-GA or vNSGA-II) provides a pool of candidate solutions that can be compared in the phenotype space according to a given distance function (e.g., Euclidian distance). According to Črepinšek et al. [65], in multi-objective optimization a widely-used approach to measure the phenotype diversity of a given solution X consists of computing the average normalized euclidean distance between X and all the other solutions within a population in the phenotype space. Specifically, let $P = \{X_1, X_2, \ldots, X_m\}$ be the set of solutions in a given population and let $F = \{f_1, \ldots, f_r\}$ the set of objective functions which define the phenotype space. The phenotype diversity of a generic solution $X_i \in P$ can be computed as follows [65]:

$$d_f(X_i) = \frac{1}{m-1} \sum_{j=1, j \neq i}^{m} \sqrt{\sum_{t=1}^{r} \left(\frac{f_r(X_i) - f_r(X_j)}{f_r^{max} - f_r^{min}} \right)^2}, \qquad (10)$$

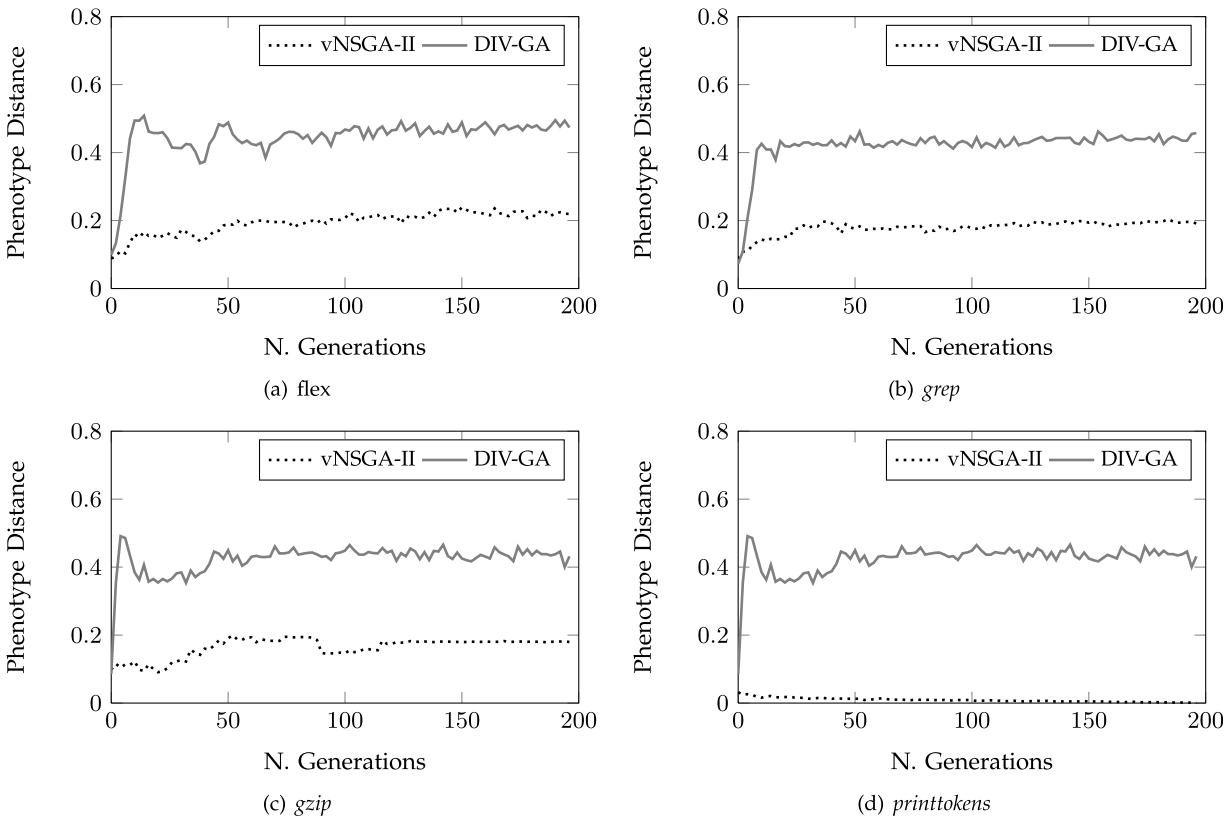
where $f_r(X_i)$ denotes the value of the objective function f_r for X_i, while f_r^{max} and f_r^{min} denote the maximum and minimum value of f_r in the current population, respectively. Thus, the phenotype diversity of a whole population can be computed as the average phenotype diversity of all its solutions:

$$d_f(P) = \frac{1}{m} \sum_{j=1}^{m} d_f(X_j). \qquad (11)$$

We can use $d_f(P)$ to measure the average phenotype diversity for each generation and compare the phenotype diversity of the populations obtained by vNSGA-II and DIV-GA at varying number of generations. To this aim, Fig. 9 plots—for some programs considered in our study, namely `flex`, `grep`,

TABLE 8
Average Execution Time for Algorithms

Program	2-Objectives			3-Objectives		
	Add. Greedy	DIV-GA	vNSGA-II	Add. Greedy	DIV-GA	vNSGA-II
bash	40 min 26 s	**18 min 41 s**	23 min 40 s	40 min 31 s	**20 min 25 s**	25 min 47 s
flex	**1 min 1 s**	2 min 45 s	4 min 21 s	**1 min 7 s**	2 min 51 s	5 min 40 s
grep	**1 min 53 s**	3 min 25 s	4 min 52 s	**2 min 4 s**	3 min 45 s	5 min 57 s
gzip	**3 s**	31 s	1 min 15 s	**4 s**	33 s	1 min 22 s
printtokens	**1 min 9 s**	2 min 59 s	10 min 27 s	**1 min 10 s**	3 min 52 s	11 min 7 s
printtokens2	**1 min 14 s**	1 min 25 s	3 min 42 s	**1 min 30 s**	1 min 32 s	4 min 8 s
schedule	**11 s**	43 s	2 min 51 s	**33 s**	1 min 15 s	4 min 23 s
schedule2	**10 s**	47 s	3 min 45 s	**31 s**	55 s	4 min 18 s
sed	**10 s**	1 min 2 s	2 min 5 s	**15 s**	1 min 33 s	2 min 25 s
space	52 s	**48 s**	2 min 11	**55 s**	1 min 7 s	2 min 20 s
vim	40 min 11 s	**17 min 59 s**	23 min 27 s	40 min 12 s	**22 min 9 s**	30 min 45 s



(a) flex (b) *grep* (c) *gzip* (d) *printtokens*

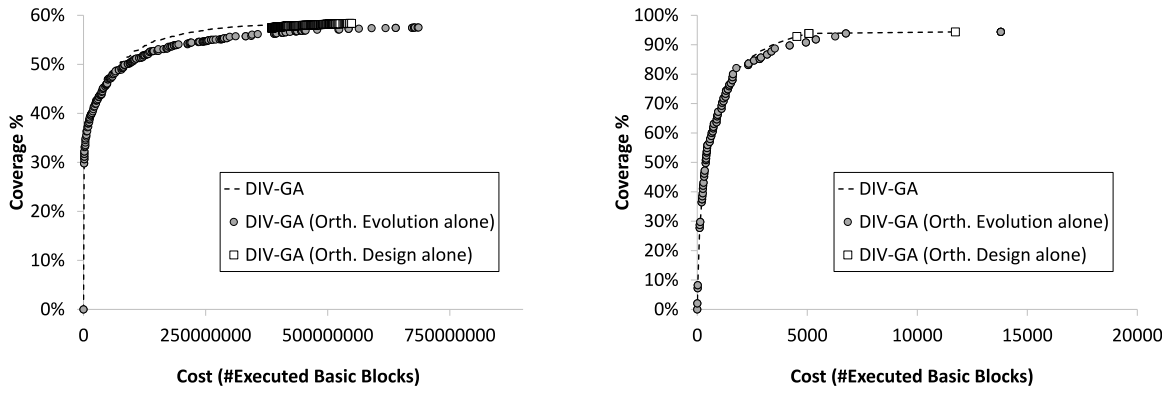
Fig. 9. Average phenotype distance between the solutions obtained by vNSGA-II and DIV-GA.

`gzip` and `printtokens`—the phenotype diversity produced by vNSGA-II and DIV-GA at different generations for the two objective formulation of the test case selection problems. We obtained consistent results for all other programs and for three-objective test case selection problem as well (see the on-line Appendix [51] for further details).

As it can be noticed, the average phenotype distance between solutions obtained by DIV-GA is always higher than those achieved by vNSGA-II. In particular, for all programs the phenotype distance between the solutions generated by DIV-GA increases within very few generations and remains greater or equal to 40 percent, while for vNSGA-II the average phenotype distance between the solutions is always lower than 22 percent, but for `printtokens` where it is lower than 5 percent. Thus, we can conclude that injecting diversity in the genotype space through orthogonal design and orthogonal evolution also increases the phenotype diversity.

6.3 Individual Contribution of Orthogonal Design and Orthogonal Evolution

Another important aspect to be further investigated is represented by the individual contribution of *orthogonal design* and *orthogonal evolution* on the performances of DIV-GA. To this aim, Fig. 10 shows—for `bash` and `printtokens`—the Pareto fronts achieved by two variants of DIV-GA: (i) DIV-GA without orthogonal evolution and (ii) DIV-GA without orthogonal design. We obtained consistent results for all other programs and for three-objective test case selection problem as well.



(a) bash (b) printtokens

Fig. 10. DIV-GA without *orthogonal design* (i) *vs.* DIV-GA without orthogonal evolution (ii).

The DIV-GA without *orthogonal design* (i.e., with orthogonal evolution alone) achieved wider Pareto fronts that are close to the fronts obtained by the complete DIV-GA (i.e., the version that includes both orthogonal evolution and orthogonal design). However, often this variant of DIV-GA fails to find optimal solutions having high levels of coverage. For example, on `bash` this variant produces solutions (sub-sets of the test suite) that are dominated by the solutions achieved by DIV-GA and the other variant with orthogonal design alone. *Vice versa*, the variant of DIV-GA which incorporates only the orthogonal design (i.e., without orthogonal evolution) on the one hand turned out to be very efficient in finding solutions having higher coverage values (such solutions are close to the solution obtained by the complete version of DIV-GA). On the other hand, such a variant is not able to find any solution with lower coverage values. Thus, we can conclude that even if both the two mechanisms introduced in this paper, i.e. orthogonal design and orthogonal evolution, are designed to increase the diversity between the obtained solutions in the *genotype space*, they have a different effects on the performances of MOGAs. Therefore, DIV-GA requires both diversity-preserving mechanisms in order to achieve good performances.

7 Threats to Validity

This section discusses the threats to the validity of our empirical evaluation, classifying them into *construct*, *internal*, *external*, and *conclusion* validity.

Threats to *construct validity* concern the relationship between theory and observation. In this study, they are mainly related to the choice of the metrics used to evaluate the characteristics of the different test case selection algorithms. In order to evaluate the optimality of the experimented algorithms (DIV-GA, additional greedy, and vNSGA-II) we used two well-known metrics: (i) Pareto frontier size, and (ii) number of solutions not dominated by reference Pareto frontiers [16]. Such metrics have been also used in previous work on multi-objective test case selection [14], [28], [71], [72]. The effectiveness of the Pareto sets achieved by all the algorithms was measured by using the hypervolume indicator, which is widely used in multi-objective optimization [4]. In particular, we used (i) execution cost and (ii) percentage of detected faults as utility functions to build a bi-objective hypervolume indicator. Another construct validity threat involves the correctness of the measures used as test criteria: statement coverage, faults coverage and execution cost. To mitigate such a threat, the code coverage information were collected using two open-source profiler/compiler tools (GNU `gcc` and `gcov`). The execution cost was measured by counting the number of source code blocks expected to be executed by the test cases, while the original fault coverage information was extracted from the SIR dataset [36].

Threats to *internal validity* concern factors that could have influenced our results and that were not properly considered. In this study, a crucial factor is the random nature of the GAs themselves [3]. To address this problem, we ran the experimented GAs (DIV-GA and vNSGA-II) 30 times for each subject program (as done in previous work [14], [71], [73]), and considered the mean values of the measures used to evaluate the optimality and effectiveness. Another threat to internal validity is represented by the algorithms used to compute orthogonal vectors in the DIV-GA: there is no unique algorithm to generate orthogonal vectors and different algorithms might affect the performance of the proposed algorithm. To address such a potential issue we report the algorithm used in this paper. The tuning of the GAs parameters is another factor that can affect the internal validity of this work. In this study we used the same parameters used in previous work on multi-objective test case selection [14], [71], [73].

Threats to *external validity* concern the generalization of our findings, and are related to the set of programs used in the experimentation. We considered 11 programs from the SIR, that were also used in most previous work on regression testing [9], [14], [37], [56], [57], [67], [71], [72], [73]. However, in order to corroborate our findings, replications on a wider range of programs and optimization techniques are desirable. The replication of the study we conducted in this paper is part of our agenda for future work. Also, there may be optimization algorithms or formulations of the test case selection problem not considered in this study that could produce better results. No particular algorithm is known to be effective for the multi-objective test case selection problem [72], and usually the evaluation of a search-based algorithm involves a comparison with other kinds of algorithms. In this paper we compared DIV-GA with the additional greedy algorithm, and vNSGA-II in order to evaluate the benefits of the proposed algorithms over the most used ones. Moreover, in order to make more generalizable the results, we evaluated all the algorithms with respect to solving two different formulations of the test case selection problem with two and three objectives to be optimized.

Finally, for what concerns *conclusion validity*, we support our findings by using appropriate statistical tests, i.e. the Welch's t-test. We performed Wilk-Shapiro normality test to verify wether the Welch's t-test could be applied to our data. Finally, we used the Cohen's d effect size to measure the magnitude of the differences between the experimented algorithms.

8 Conclusion and Future Work

This paper proposed a novel diversity-preserving technique based on orthogonal design and orthogonal evolution to improve the performance of multi-objective genetic algorithms when solving multi-criteria regression testing problems. Specifically, we proposed DIVersity based Genetic Algorithm (DIV-GA), a novel multi-objective GA which combine the main loop of the popular NSGA-II with the diversity-preserving mechanism formulated in this paper for multi-objective test case selection.

An empirical study conducted on 11 open source programs and test suites shows that DIV-GA outperforms both additional greedy algorithm and the island version of NSGA-II referred as vNSGA-II, which were considered as the best optimizers for multi-objective test case selection problem [9], [29], [50], [69], [71], [73]. In particular, DIV-GA allows not only to generate more optimal trade-offs with respect to the other optimizers when considering two and three test case selection criteria, but its selected sub-test suites turned out to be more cost-effective. Indeed, the sub-test

suites generated by DIV-GA are able to reveal more faults at same level of execution cost than the sub-test suites obtained by both the additional greedy algorithm and vNSGA-II.

The results achieved in our experimentation support our initial conjecture. Injecting diversity in the genotype space through orthogonal exploration also drastically increases diversity in the phenotype space, even when the basic MOGA already includes phenotype diversity injection mechanisms, like in the case of vNSGA-II. Without sound diversity mechanisms the potential of GAs can be seriously undermined, as shown in our empirical study. This is true in the test case selection problem, but also in other software engineering problems, such as test case generation [38].

An important issue that was analyzed by De Lucia et al. [15] and by Kifetew et al. [38] for single-objective GAs and has been confirmed in this study is that the cost of computing SVD is more than compensated by the higher convergence speed of DIV-GA. Indeed, on average DIV-GA halves the execution times of vNSGA-II and for large programs drastically reduces the execution times of the greedy algorithm.

Our final conjecture, that we plan to verify in the future, is that the proposed diversity mechanisms can be properly customized in order to improve the plethora of search-based approaches that have been recently proposed to support software engineering tasks (e.g., refactoring and scheduling) but that do not properly consider the diversity as an important issue.

REFERENCES

[1] H. E. Aguirre and K. Tanaka, "Selection, drift, recombination, and mutation in multiobjective evolutionary algorithms on scalable MNK-landscapes," in *Proc. 3rd Evol. Multi-Criterion Optimization*, 2005, pp. 355–369.

[2] 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.*, May 2011, pp. 1–10.

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

[4] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Theory of the hypervolume indicator: Optimal μ-distributions and the choice of the reference point," in *Proc. 10th ACM SIGEVO Workshop Found. Genetic Algorithms*, 2009, pp. 87–102.

[5] S. Bates and S. Horwitz, "Incremental program testing using program dependence graphs," in *Proc. 20th ACM SIGPLAN-SIGACT Symp. Principles Program. Language*, 1993, pp. 384–396.

[6] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *Proc. 26th Int. Conf. Softw. Eng.*, 2004, pp. 106–115.

[7] R. C. Bryce, C. J. Colbourn, and M. B. Cohen, "A framework of greedy methods for constructing interaction test suites," in *Proc. Int. Conf. Softw. Eng.*, 2005, pp. 146–155.

[8] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The art of test case diversity," *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, Jan. 2010.

[9] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," *Inf. Process. Lett.*, vol. 60, no. 3, pp. 135–141, Nov. 1996.

[10] C. A. Coello Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag, 2006.

[11] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Mahwah, NJ, USA: Lawrence Earlbaum Assoc., 1988.

[12] M. Cohen, M. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Trans. Softw. Eng.*, vol. 34, no. 5, pp. 633–650, Sep./Oct. 2008.

[13] W. J. Conover, *Practical Nonparametric Statistics*, 3rd ed. New York, NY, USA: Wiley, 1998.

[14] A. De Lucia, M. Di Penta, R. Oliveto, and A. Panichella, "On the role of diversity measures for multi-objective test case selection," in *Proc. 7th Int. Workshop Autom. Softw. Test*, 2012, pp. 145–151.

[15] A. De Lucia, M. Di Penta, R. Oliveto, and A. Panichella, "Estimating the evolution direction of populations to improve genetic algorithms," in *Proc. 14th Int. Conf. Genetic Evol. Comput. Conf.*, 2012, pp. 617–624.

[16] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: Wiley, 2001.

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

[18] H. Do, G. Rothermel, and A. Kinneer, "Empirical studies of test case prioritization in a Junit testing environment," in *15th Int. Symp. Softw. Reliability Eng.*, 2004, pp. 113–124.

[19] A. E. Eiben and C. A. Schippers, "On evolutionary exploration and exploitation," *Fundam. Inform.*, vol. 35, no. 1–4, pp. 35–50, 1998.

[20] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proc. 23rd Int. Conf. Softw. Eng.*, 2001, pp. 329–338.

[21] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," *Softw. Eng. Notes*, vol. 25, pp. 102–112, 2000.

[22] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2000, pp. 102–112.

[23] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002.

[24] K. Fischer, "A test case selection method for the validation of software maintenance modifications," in *Proc. COMPSAC*, 1977, pp. 421–426.

[25] V. Garousi and T. Varma, "A replicated survey of software testing practices in the Canadian province of Alberta: What has changed from 2004 to 2009?" *J. Syst. Softw.*, vol. 83, no. 11, pp. 2251–2262, Nov. 2010.

[26] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Reading, MA, USA: Addison-Wesley, 1989.

[27] G. Harik, "Finding multimodal solutions using restricted tournament selection," in *Proc. 6th Int. Conf. Genetic Algorithms*, 1995, pp. 24–31.

[28] M. Harman, "Making the case for MORTO: Multi objective regression test optimization," in *Proc. ICST Workshops*, 2011, pp. 111–114.

[29] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, pp. 270–285, 1993.

[30] H. Hemmati, A. Arcuri, and L. Briand, "Reducing the cost of model-based testing through test case diversity," in *Proc. 22nd IFIP WG 6.1 Int. Conf. Testing Softw. Syst.*, 2010, pp. 63–78.

[31] H. Hemmati, A. Arcuri, and L. Briand, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," in *Proc. 4th IEEE Int. Conf. Softw. Testing, Verification Validation*, 2011, pp. 327–336.

[32] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 6.1–6.42, 2013.

[33] T. Hiroyasu, K. Kobayashi, M. Nishioka, and M. Miki, "Diversity maintenance mechanism for multi-objective genetic algorithms using clustering and network inversion," in *Proc. 10th Int. Conf. Parallel Problem Solving Nature: PPSN X*, 2008, pp. 722–732.

[34] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.

[35] S. Holm, "A simple sequentially rejective Bonferroni test procedure," *Scandinavian J. Statist.*, vol. 6, pp. 65–70, 1979.

[36] S. G. E. Hyunsook Do and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empir. Softw. Eng.: An Int. J.*, vol. 10, pp. 405–435, 2005.

[37] D. Jeffrey, "Test suite reduction with selective redundancy," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2005, pp. 549–558.

[38] F. M. Kifetew, A. Panichella, A. De Lucia, R. Oliveto, and P. Tonella, "Orthogonal exploration of the search space in evolutionary test case generation," in *Proc. Int. Symp. Softw. Testing Anal.*, 2013, pp. 1–11.

[39] M. Köppen and K. Yoshida, "Substitute distance assignments in nsga-ii for handling many-objective optimization problems," in *Proc. 4th Int. Conf. Evol. Multi-Criterion Optim.*, 2007, pp. 727–741.
[40] H. Li, Y.-C. Jiao, L. Zhang, and Z.-W. Gu, "Genetic algorithm based on the orthogonal design for multidimensional knapsack problems," *Adv. Nat. Comput.*, vol. 4221, pp. 696–705, 2006.
[41] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 33, no. 4, pp. 225–237, Apr. 2007.
[42] H. Maaranen, K. Miettinen, and A. Penttinen, "On initial populations of a genetic algorithm for continuous optimization problems," *J. Global Optim.*, vol. 37, no. 3, pp. 405–436, Mar. 2007.
[43] S. W. Mahfoud, "Niching methods for genetic algorithms," Illinois Genetic Algorithms Laboratory, Univ. Illinois at Urbana-Champaign Champaign, IL, USA, Tech. Rep. No. 9500, 1995.
[44] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel, and S. Elbaum, "Cost-cognizant test case prioritization," Dept. Comput. Sci. Eng., Univ. Nebraska-Lincoln, Lincoln, Nebraska, USA, Tech. Rep. TR-UNL-CSE-2006-0004, Mar. 2006.
[45] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Struct. Multidisciplinary Optim.*, vol. 26, pp. 369–395, 2004.
[46] M. Marré and A. Bertolino, "Using spanning sets for coverage testing," *IEEE Trans. Softw. Eng.*, vol. 29, no. 11, pp. 974–984, Nov. 2003.
[47] MATLAB, *Version 7.10.0 (R2010b)*. Natick, MA, USA: The MathWorks Inc., 2010.
[48] S. Mcmaster and A. M. Memon, "Call stack coverage for test suite reduction," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2005, pp. 539–548.
[49] D. C. Montgomery, *Design amd Analysis of Experiments*, 3rd ed. New York, NY, USA: Wiley, 2009.
[50] A. J. Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proc. 12th Int. Conf. Testing Comput. Softw.*, 1995, pp. 111–123.
[51] A. Panichella, R. Oliveto, M. Di Penta, and A. De Lucia. (2013, Oct.). Improving multi-objective test case selection by injecting diversity in genetic algorithms," Dept. Manage. Inform. Technol., University of Salerno, Fisciano Salerno, Italy, Tech. Rep. [Online]. Available: http://www.sesa.dmi.unisa.it/reports/DIVGA/
[52] G. Rothermel and M. Harrold, "A safe, efficient algorithm for regression test selection," in *Proc. Conf. Softw. Maintenance*, 1993, pp. 358–367.
[53] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.
[54] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Trans. Softw. Eng.*, vol. 22, no. 8, pp. 529–551, Aug. 1996.
[55] G. Rothermel and M. J. Harrold, "Empirical studies of a safe regression test selection technique," *IEEE Trans. Softw. Eng.*, vol. 24, no. 6, pp. 401–419, Jun. 1998.
[56] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proc. Int. Conf. Softw. Maintenance*, 1998, pp. 34–44.
[57] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *J. Softw. Testing, Verification, Rel.*, vol. 12, pp. 219–249, 2002.
[58] P. Runeson, "A Survey of Unit Testing Practices," *IEEE Softw.*, vol. 23, no. 4, pp. 22–29, Jul./Aug. 2006.
[59] S. Sampath, R. Bryce, and A. Memon, "A uniform representation of hybrid criteria for regression testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1326–1344, Oct. 2013.
[60] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *Proc. Int. Symp. Empir. Softw. Eng.*, 2005, pp. 60–63.
[61] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," in *Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2002, pp. 97–106.
[62] D. R. Stinson, "Combinatorial designs: Constructions and analysis," *SIGACT News*, vol. 39, no. 4, pp. 17–21, 2008.
[63] G. Strang, *Introduction to Linear Algebra*, 4th ed. Cambridge, MA, USA: Wellesley-Cambridge, 2009.
[64] A. Toffolo and E. Benini, "Genetic diversity as an objective in multi-objective evolutionary algorithms," *Evol. Comput.*, vol. 11, pp. 151–167, 2003.
[65] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 35:1–35:33, 2013.
[66] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time aware test suite prioritization," in *Proc. Int. Symp. Softw. Testing Anal.*, 2006, pp. 1–12.
[67] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in *Proc. 17th Int. Conf. Softw. Eng.*, 1995, pp. 41–50.
[68] S. S. Yau and Z. Kishimoto, "A method for revalidating modified programs in the maintenance phase," in *Proc. Int. Comput. Softw. Appl. Conf.*, 1987.
[69] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verif. Rel.*, vol. 22, no. 2, pp. 67–120, Mar. 2012.
[70] S. Yoo, "A novel mask-coding representation for set cover problems with applications in test suite minimisation," in *Proc. 2nd Int. Symp. Search-Based Softw. Eng.*, 2010, pp. 19–28.
[71] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proc. ACM/SIGSOFT Int. Symp. Softw. Testing Anal.*, 2007, pp. 140–150.
[72] S. Yoo and M. Harman, "Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation," *J. Syst. Softw.*, vol. 83, no. 4, pp. 689–701, 2010.
[73] S. Yoo, M. Harman, and S. Ur, "Highly scalable multi objective test suite minimisation using graphics cards," in *Proc. 3rd Int. Conf. Search Based Softw. Eng.*, 2011, pp. 219–236.
[74] Q. Zhang and Y.-W. Leung, "An orthogonal genetic algorithm for multimedia multicast routing," *IEEE Trans. Evol. Comput.*, vol. 3, no. 1, pp. 53–62, Apr. 1999.
[75] J. Zhu, G. Dai, and L. Mo, "A cluster-based orthogonal multiobjective genetic algorithm," *Comput. Intell. Intell. Syst.*, vol. 51, pp. 45–55, 2009.
[76] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration," in *Proc. 4th Int. Conf. Evol. Multi-Criterion Optim.*, 2007, pp. 862–876.

Annibale Panichella received the MSc degree in computer science from the University of Salerno, Italy, in 2010. He received the PhD degree in software engineering from the Department of Information Technology of the University of Salerno in 2014, under the supervision of Prof. Andrea De Lucia and Prof. Rocco Oliveto. He is currently a collaborator of the Security&Trust research unit at the Center for Information Technologies of Fondazione Bruno Kessler in Trento. His research interests include software testing, search-based software engineering, evolutionary computation, information retrieval, defect prediction, security testing, and empirical software engineering. He is a member of the IEEE.

Rocco Oliveto is an assistant professor in the Department of Bioscience and Territory at University of Molise, Italy. He is the chair of the Computer Science Program and the director of the Laboratory of Computer Science and Scientific Computation of the University of Molise. He received the PhD in Computer Science from University of Salerno, Italy, in 2008. His research interests include traceability management, information retrieval, software maintenance and evolution, search-based software engineering, and empirical software engineering. He is an author of about 100 papers appeared in international journals, conferences and workshops. He serves and has served as organizing and program committee member of international conferences in the field of software engineering. He is a member of the IEEE Computer Society, IEEE, and ACM.

Massimiliano Di Penta is an associate professor at the University of Sannio, Italy. His research interests include software maintenance and evolution, mining software repositories, empirical software engineering, search-based software engineering, and service-centric software engineering. He is an author of more than 190 papers appeared in international journals, conferences and workshops. He serves and has served in the organizing and program committees of more than 100 conferences such as ICSE, FSE, ASE, ICSM, ICPC, GECCO, MSR WCRE, and others. He has been a general co-chair of various events, including the 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM 2010), the second International Symposium on Search-Based Software Engineering (SSBSE 2010), and the 15th Working Conference on Reverse Engineering (WCRE 2008). Also, he has been program chair of events such as the 28th IEEE International Conference on Software Maintenance (ICSM 2012), the 21st IEEE International Conference on Program Comprehension (ICPC 2013), the ninth and 10th Working Conference on Mining Software Repository (MSR 2013 and 2012), the 13th and 14th Working Conference on Reverse Engineering (WCRE 2006 and 2007), the first International Symposium on Search-Based Software Engineering (SSBSE 2009), and other workshops. He is currently member of the steering committee of ICSME, MSR, SSBSE, and PROMISE. Previously, he has been a steering committee member of other conferences, including ICPC, SCAM, and WCRE. He is in the editorial board of the *IEEE Transactions on Software Engineering*, the *Empirical Software Engineering Journal* edited by Springer, and of the *Journal of Software: Evolution and Processes* edited by Wiley. He is a member of the IEEE.

Andrea De Lucia received the PhD degree in electronic engineering and computer science from the University of Naples "Federico II," Italy, in 1996. He is a full professor of software engineering at the Department of Management & Information Technology of the University of Salerno, Italy, the head of the Software Engineering Lab, and the director of the International Summer School on Software Engineering. His research interests include software maintenance and testing, reverse engineering and reengineering, empirical software engineering, search-based software engineering, collaborative development, workflow and document management, and e-learning. He has published more than 200 papers on these topics in international journals, books, and conference proceedings and has edited books and journal special issues. He also serves on the editorial boards of international journals and on the organizing and program committees of international conferences. He is a senior member of the IEEE and the IEEE Computer Society and was also at-large member of the executive committee of the IEEE Technical Council on Software Engineering (TCSE).