Many-Objective Test Suite Generation for Software Product Lines

ROBERT M. HIERONS, The University of Sheffield
MIQING LI, The University of Birmingham
XIAOHUI LIU, Brunel University
JOSE ANTONIO PAREJO and SERGIO SEGURA, Universidad de Sevilla
XIN YAO, Southern University of Science and Technology and The University of Birmingham

A Software Product Line (SPL) is a set of products built from a number of features, the set of valid products being defined by a feature model. Typically, it does not make sense to test all products defined by an SPL and one instead chooses a set of products to test (test selection) and, ideally, derives a good order in which to test them (test prioritisation). Since one cannot know in advance which products will reveal faults, test selection and prioritisation are normally based on objective functions that are known to relate to likely effectiveness or cost. This article introduces a new technique, the grid-based evolution strategy (GrES), which considers several objective functions that assess a selection or prioritisation and aims to optimise on all of these. The problem is thus a many-objective optimisation problem. We use a new approach, in which all of the objective functions are considered but one (pairwise coverage) is seen as the most important. We also derive a novel evolution strategy based on domain knowledge. The results of the evaluation, on randomly generated and realistic feature models, were promising, with GrES outperforming previously proposed techniques and a range of many-objective optimisation algorithms.

Additional Key Words and Phrases: Software product line, test selection, test prioritisation, multi-objective optimisation

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT projects BELI (TIN2015-70560-R) and HORATIO (RTI2018-101204-B-C21), the Science and Technology Innovation Committee Foundation of Shenzhen (ZDSYS201703031748284), Shenzhen Peacock Plan (KQTD2016112514355531), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), and EPSRC (EP/J017515/1 and EP/P005578/1).
Authors' addresses: R. M. Hierons, The University of Sheffield; email: r.hierons@sheffield.ac.uk; M. Li, The University of Birmingham; email: m.li.8@cs.bham.ac.uk; X. Liu, Brunel University; email: XiaoHui.Liu@brunel.ac.uk; J. A. Parejo and S. Segura, Universidad de Sevilla; emails: {japarejo, sergiosegura}@us.es; X. Yao, Southern University of Science and Technology and The University of Birmingham; email: xiny@sustc.edu.cn.

1 INTRODUCTION

A Software Product Line (SPL) is a set of products, which are built from a number of features, a feature being some aspect of functionality [11]. A product can be seen as being a set of features and feature models are often used to define the allowed combinations of features and so the valid products. There is evidence of a number of companies using feature models, including Boeing [64], Siemens [35], and Toshiba [55].

Ideally one would test all of the products in an SPL defined by a feature model. However, a feature model might specify a vast number of products; often it is not feasible to test every valid product. In addition, since the products of an SPL are built from a common set of features, one would expect that the testing of all valid products would introduce significant redundancy. Thus, even if one could test all valid products, this is likely to be inefficient and, assuming a limited testing budget, might also be ineffective. These observations have led to two problems being investigated: the problem of choosing a set of products to test (the *test selection problem*) and the problem of choosing a good order in which products should be tested (the *test prioritisation problem*). Test selection aims to choose products that are likely to reveal any faults that are present, while prioritisation aims to find faults as early as possible. Previous work in this area has taken the view that these problems complement each other and has therefore considered selection and prioritisation together [24, 29, 50, 73]. This is the approach we also take in this article.

The selection and prioritisation problems, as described above, are expressed in terms of the actual faults in the products of an SPL. However, normally the tester does not have access to this information before testing begins. Thus, the problem is to find a test suite (sequence of valid products) that is *likely* to be a good solution (in terms of selection and prioritisation). It has been observed that a number of properties of good test suites can be captured by *objective functions* that map a test suite to a value that represents how "good" this test suite is according to the properties (see, for example, [19], [23], [29], [31], [32], [48], [50], [57], [60], [70], [71], and [73]). For example, a fault might be associated with the interaction of a pair of features and so we might want to test as many such interactions as possible (pairwise coverage). This is an example of a functional objective; one associated with the functionality of the features and how the features are combined in the feature model. There are also non-functional objectives (quality attributes) such as the cost of a feature.

Since a number of objectives/fitness functions have been identified, the problem of finding a good solution has been expressed as a multi-objective optimisation problem [19, 23, 24, 31, 47, 48, 54, 57, 70, 71, 73]. Previous work developed test-generation approaches based on evolutionary multi-objective algorithms and found these to be effective [19, 23, 31, 48, 57, 70, 71, 73]; a recent survey was provided by Lopez-Herrejon et al. [49].

The literature on test generation from feature models has three important limitations that we address in this article. First, most previous work has looked at multi-objective optimisation problems, in which there are two or three objectives, but only a few have explored the benefits of many-objective optimisation, with four [23, 73] and five objectives [71]. In fact, Lopez-Herrejon et al. [49] explicitly mentioned many-objective search in SPL testing as an open challenge. Second, previous work has typically addressed the problem by either weighting the objectives or separately considering test selection and prioritisation. Third, all previous approaches have resorted to standard state-of-the-art optimisation algorithms neglecting the potential benefits of exploiting the knowledge of the problem to develop more powerful algorithms. As a result, such approaches may either be unable to provide a set of good trade-offs for the tester or fail to guarantee the quality of the test suite on some objectives.

In this article, we address the many-objective optimisation problem and devise a novel approach to this optimisation problem, *the grid-based evolution strategy (GrES)*. In addition, the proposed

approach is inspired by the observation that typically, when testing the products of a product line, the tester will want to ensure that all pairs of features are tested. The motivation for this is that combinatorial testing, and specifically pairwise testing, is the main coverage criterion considered in the SPL literature (e.g., in [52], the authors identified over 40 approaches for combinatorial testing in SPL). The essential idea is that we have a type of coverage that should be achieved and we want the "best" solution that achieves this coverage. Naturally, it would be straightforward to generalise the approach to the case where one wants to achieve a certain level of pairwise coverage that is less than 100%. As a result of our approach, the proposed evolution strategy prioritises pairwise coverage: when comparing two solutions, it first compares them on pairwise coverage and only then (if they have the same pairwise coverage) on the other objectives. In Section 3, we further explain and justify this decision.

An alternative approach would be to weight the different objectives, in order to turn a multi-objective optimisation problem into a single-objective optimisation problem (see, for example, [19], [31], [70], [73]). However, the multi-objective approach has a number of advantages. In the single-objective approach, when giving a high weight to a particular objective (such as pairwise coverage), one is making this "more important" but by a fixed multiplier. Thus, for example, if we have two objectives and we give the first a weighting ten times that of the second, then (under minimisation) we prefer (3,12) to (4,3) but we do not prefer (3,12) to (4,1). In contrast, in our approach, we always prefer a solution with higher pairwise coverage—we are saying that pairwise coverage is more important than the other objectives, rather than it being X times more important for some fixed X. This better captures the nature of the problem. In addition, the multi-objective approach provides a range of trade-offs between the other objectives (those other than pairwise coverage) and the developer can then assess these alternatives and choose from them. Note also that GrES does not require the developer to choose values for weights; for the single-objective approach, it is unclear how a developer would choose weights without, for example, some initial experiments/tuning.

This article makes the following main contributions. It proposes a novel approach to many-objective test suite generation for software product lines, in which we consider nine fitness functions. Novelty is also provided by the algorithm design, which uses a rarely used class of evolutionary algorithms (evolution strategy) that works on a small population and does not have a crossover operation (i.e., asexual reproduction—variance being introduced only by mutation). We also customised the evolution strategy by taking advantage of domain knowledge in selecting individuals for variation. This article then reports on the results of experiments that compared GrES with two algorithms previously devised for this problem and a range of multi-objective optimisation algorithms. The evaluation used randomly generated feature models, in order to enable us to explore the effect of the size of the feature model. It also used 19 realistic feature models: feature models from the literature with randomly generated attribute values. Finally, this article reports on the results of an experiment using a realistic feature model with real attribute values.

In the experiments, we found that the proposed approach for comparing individuals always led to a population in which all individuals (test suites) provided full pairwise coverage; in contrast, the other approaches returned populations in which at most 10% of individuals provided full pairwise coverage. In addition, GrES returned populations that had higher quality, as assessed by their hyper-volume (HV)[1] [81], with many of the effect sizes being large. In order to evaluate whether the use of an evolution strategy was a good choice, we also had experiments in which a number of different multi-objective approaches were adapted to use our novel comparison mechanism. In these experiments, all approaches returned a population whose individuals all had full pairwise

[1]HV is a metric which can reflect the quality of a solution set in both approaching the optimal solution set and diversifying its solutions.

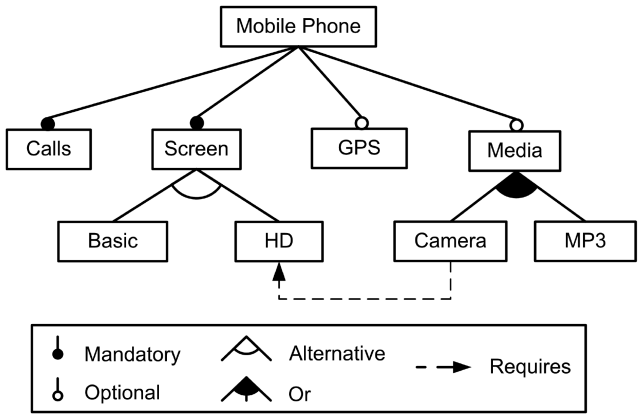

Fig. 1. Mobile phone feature model [57].

coverage. However, the evolution strategies returned the highest HV values, with most differences being statistically significant. Two previous multi-objective approaches used variants of NSGA-II but with fewer objectives [48, 57]. We had experiments in which we compared GrES with these two approaches, restricting the objectives considered in the evaluation to those previously used. Unsurprisingly, the results more mixed: GrES produced lower HV values for the smaller models but higher HV values for the larger models. Note that we did not change the set of objectives that GrES was attempting to optimise and so GrES had the disadvantage of considering additional objectives that were not used in the comparison. Finally, experiments were performed to assess scalability; in this it was found that GrES is faster than the two previous approaches and that the approaches that use our novel comparison mechanism had similar execution times.

This article is structured as follows: In Section 2, we review background material, and in Section 3, we describe our novel many-objective approach. Section 4 then outlines the experimental design while Section 5 gives the results of the experiments and discusses these. In Section 6, we discuss threats to validity and Section 7 then places our work in the context of the literature. Finally, in Section 8, we draw conclusions and discuss possible lines of future work.

2 BACKGROUND

2.1 Feature Models

Software Product Lines (SPL) engineering focuses on the systematic development of families of software products [11]. Products in SPLs are represented in terms of features where a feature is any distinguishable characteristic relevant for the stakeholders [37]. A *feature model* is a tree-like representation of all the products of an SPL in terms of features and relationships among them. These relationships constrain the way in which features can be combined to form valid products. Feature models were first introduced as a part of the Feature-Oriented Domain Analysis method (FODA) by Kang et al. back in 1990 [37]. Since then, feature modelling has become the de-facto standard for variability management in SPLs. Figure 1 depicts a sample feature model representing an SPL for mobile phones [57]. The software loaded into the mobile phone is determined by the features that it supports, e.g., Camera. The root feature ("Mobile Phone") identifies the SPL. The following types of relationships constrain how features can be combined in a product:

- *Mandatory.* If a feature has a mandatory relationship with its parent feature, it must be included in all the products in which its parent feature appears. In Figure 1, all the products of the SPL must provide support for the mandatory feature `Calls`.

Table 1. Mobile Phone Feature Attributes

Feature	Changes	Cost	Faults	Size
Basic	1	25	0	270
Calls	6	20	5	1,000
Camera	12	35	7	680
GPS	8	45	11	460
HD	3	25	2	510
Media	9	5	5	1,100
MP3	4	10	3	390
Screen	2	5	1	930

- *Optional.* If a feature has an optional relationship with its parent feature, it can be optionally included in the products that include its parent feature. For instance, GPS is defined as an optional feature in Figure 1.
- *Alternative.* The children of a feature are defined as alternatives if exactly one feature should be selected when its parent feature is part of the product. In the example, mobile phones must support a Basic or a High Definition (HD) screen, but not both in the same product.
- *Or-Relation.* Child features are said to have an or-relation with their parent when one or more of them can be included in the products in which its parent feature appears. In Figure 1, mobile phones may include a Camera, an MP3 player, or both of them.

In addition to hierarchical relationships, feature models can also contain Cross-Tree Constraints (CTCs) between features. These are typically of the form "Feature A requires feature B" or "Feature A excludes feature B". In Figure 1, mobile phones including a Camera require an HD screen.

Feature models can be extended with extra-functional information by means of *feature attributes*. These models are referred to as *attributed feature models*. An attribute usually consists of a name, a domain, and a value, e.g., GPS.cost = 20. Attributes are usually represented either graphically in the feature model or as a table indicating the values of the attributes for each feature. Table 1 depicts four sample numeric attributes and random values for the features of the model in Figure 1, namely *Changes* (number of changes), *Cost* (estimated cost of the resources needed to test the feature), *Faults* (number of reported defects), and *Size* (number of lines of code).

Feature models can be automatically analysed to extract information from them. Catalogues with up to 30 analysis operations on feature models have been published [9]. Typical analysis operations allow us to know whether a feature model is consistent (it represents at least one product), the number of products represented by a feature model, or whether a feature model contains any errors. The analysis of feature models is supported by a number of tools including the *FaMa* framework [68], FeatureIDE [67], PLEDGE [30], and Clafer [8].

SPL testing approaches typically follow a model-based approach, where the feature model represents the testing space of the product line, and the products represent potential test cases. In this article, a test case is a product and so we use these two terms interchangeably. Likewise, we define a test suite to be a sequence of test cases, i.e., products. The number of products represented by a feature model tends to increase exponentially with the number of features. Hence, for instance, the Drupal feature model, derived from the popular Web framework Drupal, has 48 features and 21 cross-tree constraints, and it represents $2.09 \cdot 10^9$ potential products [59]. Thus, typically it is not feasible to test every single product. To address this problem, several test case selection techniques have been proposed to select a representative subset of the products to be tested. Among them, pairwise testing is a highly used technique that selects test suites that contain all the possible

Table 2. Mobile Phone Pairwise Test Suite with Pairwise Coverage of Features

ID	Test Case
TC1	Mobile Phone, Calls, Screen, Basic, Media, MP3
TC2	Mobile Phone, Calls, Screen, HD, GPS, Media, Camera, MP3
TC3	Mobile Phone, Calls, Screen, HD, Media, Camera
TC4	Mobile Phone, Calls, Screen, HD
TC5	Mobile Phone, Calls, Screen, Basic, GPS

combinations of pairs of features [52]. For instance, Table 2 depicts a pairwise suite for the model in Figure 1, where the number of products is reduced from 12 (total number of products represented by the model) to 5 containing all the possible feature pairs. However, as will be explained later, there are many other properties of a good selection or prioritisation that can be captured by a range of fitness functions. For example, recently changed features tend to be more fault prone and so ideally should be included in products that are tested relatively early in the process.

2.2 Evolutionary Multi-Objective Optimisation

In multi-objective optimisation, we have two or more objectives/fitness functions that we wish to optimise. Let us suppose that we have a multi-objective optimisation problem in which the search space is X. The classic approach to comparing two candidate solutions $A \in X$ and $B \in X$ is to use *Pareto dominance* [12], where A is said to Pareto dominate B if A is better than B on at least one objective and at least as good as B on all other objectives. Thus, if A Pareto dominates B and we have already found A, then there is no point in considering B; A is preferable to B. In contrast, if A and B are not related under Pareto dominance then they represent different trade-offs between the objectives. Under Pareto dominance, the "best" candidate solutions are those in the *Pareto optimal set* (called *Pareto front*) in the objective space, which is the set of elements in X that are not Pareto dominated by any other element in X.

Evolutionary algorithms (EAs) are a class of stochastic search and optimisation methods that mimic the process of natural evolution. Over the past two decades, there has been significant interest in the use of EAs to solve multi-objective optimisation problems (MOPs) [12, 16, 79], the resulting research branch called *evolutionary multi-objective optimisation (EMO)*. The success of EMOs can be attributed to two major advantages of EAs. One is that they have low requirements on the problem characteristics and are capable of handling large and highly complex search spaces. The other is that their population-based search can achieve an approximation of the problem's whole Pareto front. The population returned represents a set of alternative trade-offs between the objectives.

In general, evolutionary algorithms can be loosely categorised into three metaheuristics: genetic algorithm (GA), evolution strategy (ES), and genetic programming (GP) [4]. GA and ES are commonly used to generate solutions to optimisation and search problems, while the solutions in GP are in the form of computer programs. GA often operates on a relative large population (e.g., with 100 individuals), and performs both crossover and mutation to produce new individuals. In contrast, ES is based on ideas of self-adaptation, and often has a very small population and no crossover in its variation operation [62]. Formally, an ES has a $(\mu + \lambda)$ evolution form, where μ and λ are the number of parental individuals and offspring individuals, respectively. In EMO, most algorithms are GAs; very few are based on the ES principles and the most popular multi-objective ES is the Pareto archived evolution strategy (PAES) [38, 39].

Many-objective optimisation refers to the simultaneous optimisation of four or more objectives [40]. There are a number of differences between many-objective optimisation and multi-objective optimisation with two or three objectives. The increase in objective dimensionality can bring many difficulties, such as the ineffectiveness of the Pareto dominance criterion, aggravation of the conflict between convergence and diversity, inaccuracy of density estimation, inefficiency of recombination operation, increasing sensitivity to parameter settings, and rapid increase of time or space requirement. These issues cause significant challenges when using EAs in dealing with many-objective optimisation problems, especially for Pareto-based approaches.

3 THE PROPOSED ALGORITHM

In this section, we describe the proposed algorithm, the grid-based evolution strategy (GrES). We start by presenting an approach to optimisation (i.e., how to view the objectives to be optimised) that takes into account domain knowledge. We then explain the motivation for using an evolution strategy (rather than a generic genetic algorithm), and also what makes GrES different from existing evolution strategies. Next we describe the method used for selecting parental individuals for variation in the evolutionary process. This is followed by an explanation of the individual encoding and variation operators. Finally, we give the main procedure of GrES.

3.1 Optimisation Approach

As stated previously, SPL test suite generation can be seen as a combination of two problems, the test selection problem and the prioritisation problem. Test selection aims to choose products (as few as possible) that are likely to reveal any faults that are present, while prioritisation aims to find faults as early as possible. These problems result in two groups of objectives to be optimised in test suite generation: selection (including pairwise coverage), and prioritisation. Note that the two groups often have several objectives, and this thus forms an optimisation problem with *many* objectives.

Much of the previous work in the area has integrated all (or some) of the above objectives into one single fitness function with weighting factors (see, for example, [31], [54], [70], [73]). As previously discussed, when giving a high weight to a particular objective (such as pairwise coverage), one is making this more important by a fixed multiplier. This less effectively represents the nature of the problem, in which one objective is more important than the others (and not by a fixed ratio).

Another (complementary) approach that has been used is to separately consider test selection and prioritisation (see, for example, [1], [19], [29], [31], [47], [48], [54], [57], [60]). These approaches can decrease the difficulty of the considered problem from many objectives to multiple objectives. However, they may either be unable to provide a set of good trade-offs for the tester or fail to guarantee the quality of the test suite on some objectives (i.e., those not being considered in the optimisation).

In this article, we use nine fitness functions that represent both groups of objectives and aim to achieve a set of good trade-offs between them. Yet, we do not treat all the objectives equally; instead we consider pairwise coverage first and then optimise the remaining objectives simultaneously. This was motivated by the fact that pairwise coverage can be seen as the main aim in test suite generation—we want to achieve full pairwise coverage and ideally also do well for other objectives. Another advantage of this approach is that it avoids the simultaneous optimisation of heavily conflicting objectives (i.e., pairwise coverage versus cost), thus enabling Pareto-based approaches (those using Pareto dominance to assign individuals' fitness) to work well on this many-objective problem. This can be seen in the proposed Pareto-based algorithm GrES.

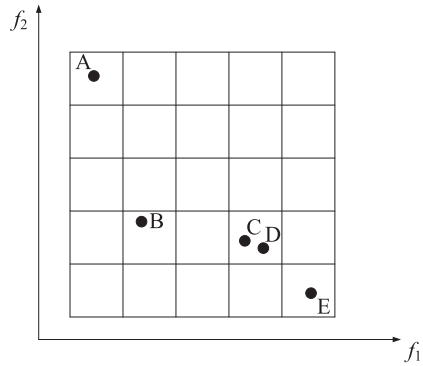

Fig. 2. Grid-based archive in a bi-objective case, where the grid degree of nondominated solutions A, B, C, D and E is 1, 1, 2, 2 and 1, respectively. For selection (i.e., *Selection_III*(Q) in Step 3 of Algorithm 1), only the solution with the lowest grid degree (A, B or E) is considered. For archive truncation (i.e., Algorithm 5), only the solution with the highest grid degree (C or D) is considered. The grid environment was constructed by predefined divisions and the boundaries of the nondominated solutions adaptively, see [76] for details.

Specifically, we use two simple rules to compare two individuals in the (environmental) selection process of GrES[2]: (1) preferring the individual with higher pairwise coverage and (2) preferring the individual with better fitness (determined by the other objectives) when their pairwise coverage is equal.

3.2 Why Use an Evolution Strategy

We ran some initial experiments with EAs on the considered SPL testing problem, with a seeding strategy that used the CASA tool to generate an initial test suite that achieves full pairwise coverage, as in [48]. In these initial experiments, we observed that, with the seeding strategy, evolutionary search that had no crossover operation performed better than (or at least as well as) search that used crossover. And we also observed that the crossover between the seed individual and a randomly generated individual typically produced offspring with worse performance (i.e., prioritisation objectives and cost objectives) than the seed individual. One explanation for this is that a crossover operation, which typically brings a big change to the individual, could disturb some good building blocks of chromosomes in the seed individual.

In addition, we also observed that in an algorithm without crossover the individuals in the final population were all from the seed individual (i.e., all being offspring of the seed individual). This means that randomly generated individuals in the initial population make no contribution to the final result.

Given these two observations, evolution strategy, which has a very small population size and no crossover operator, seems a good option. That is, we continually do some variations around the seed individual and its offspring during the evolutionary process in an effort to produce some new promising individuals. This has the potential to significantly reduce the number of evaluations, resulting from the randomly generated individuals, and make the search focus on the seed individual.

Like PAES [39], GrES uses a grid-based archive (see Figure 2) to store promising solutions (nondominated solutions[3]) generated during the evolutionary process. However, GrES significantly

[2]These two rules can apply to any EMO algorithm, and in our experimental studies, the peer EMO algorithms used the same rules in their selection process.

[3]Note that the dominance rule is based on the proposed optimisation approach; i.e., for two solutions p and q, if p dominates q, then either p has a better coverage than q, or p has the same coverage as q but dominates q on the remaining objectives.

differs from PAES (and other ESs) in that the current population (i.e., the solutions that act as parental individuals to generate offspring) is not based on the last-generation population and their offspring, but rather its elements are selected carefully from the archive set. In each generation, GrES selects three specific individuals from the archive according to some domain knowledge (this will be explained in detail in the next subsection) to form the current population and mutates them to form three new individuals, thus resulting in a (3+3) evolution form.

Finally, it is worth mentioning that the grid-based archive of GrES is managed by an individual-centred calculation environment (inspired by [76]) whose computational cost is virtually independent of the number of hyperboxes in the grid and only increases linearly with the number of objectives. This contrasts with many grid-based EAs (such as PAES) whose selection operation entails a traversal of all the hyperboxes in the grid; the number of such hyperboxes grows exponentially with the number of objectives [14].

3.3 Selection for Variation

In an evolving population, the selection of the individuals from which to produce offspring (called mating selection) is crucial. Unlike existing ESs, whose population always comes from either the last-generation population or their offspring, GrES constructs the population by selecting some specific solutions from the archive set. Here, we explain how domain knowledge was used to design the mating selection of the algorithm. Specifically, GrES takes advantage of the fact that in the test suite generation problem the cost objectives are not directly in conflict with the prioritisation objectives. That is, removing a product in a test suite does not necessarily worsen the prioritisation objectives of the suite, and changing the order of products in a test suite for a better prioritisation does not increase the cost of the test suite. This also means that a test suite may achieve both a high prioritisation and a low cost. In light of this, we consider the following two operations.

One operation randomly performs some variations on an individual in the archive to achieve a better cost or prioritisation. For better cost, a test case removal operation (which will be explained in the next subsection) is carried out in order to tentatively reduce the suite size. For better prioritisation, a test case swap operation is carried out to give a perturbation. Note that, in each generation of GrES, only one individual in the archive set is selected to perform the above variations. The selection is based on the grid; this process is like the one in PESA-II [13]. First, we randomly select one non-empty hyperbox in the grid and then randomly select one individual from this hyperbox.

The other operation is applied to the individual having the best prioritisation under one specific size of the current test suites; i.e., from a set of test suites with the same number of test cases in the population, the operator selects one with the best prioritisation (determined by the sum of the normalised prioritisation objectives). For this individual, one of the four variations, test case swap, test case removal, test case addition and test case substitution, is randomly performed. The test case removal operation is to tentatively reduce the suite size of the individual. The test case swap and addition operations are to tentatively increase the prioritisation values of the individual. The test case substitution is for both.

Finally, as the archive only allows the entry of new individuals that are not dominated by the existing individuals in the archive, it contains the best individuals produced so far. Individuals located in sparse areas of the archive tend to have not been developed well, e.g., they are probably newly produced individuals that just entered the archive. Here, we try to explore such individuals. To do so, we consider a well-known density estimator, grid degree [13]. Specifically, we first consider a non-empty hyperbox with the fewest individuals in the archive (randomly select one if there are two or more such hyperboxes) and then randomly select one individual from this hyperbox; see an illustration in Figure 2. After the selection, one of the above four variations are randomly performed on the selected individual for a better cost or prioritisation.

ALGORITHM 1: $GrES(s, N)$

1: $Q \leftarrow \{s\}$ /* s is the seed individual */
2: **while** termination criterion not fulfilled **do**
3: $\mu_1 \leftarrow Selection\_I(Q)$, $\mu_2 \leftarrow Selection\_II(Q)$, $\mu_3 \leftarrow Selection\_III(Q)$
4: $\lambda_1 \leftarrow VariationSR(\mu_1)$, $\lambda_2 \leftarrow VariationSRAS(\mu_2)$, $\lambda_3 \leftarrow VariationSRAS(\mu_3)$
5: $Q \leftarrow Join(Q, \lambda_1)$, $Q \leftarrow Join(Q, \lambda_2)$, $Q \leftarrow Join(Q, \lambda_3)$
/* N is the capacity of the archive set */
6: **if** $|Q| > N$ **then**
7: $Q \leftarrow Truncate(Q, N)$
8: **end if**
9: **end while**
10: **return** Q

3.4 Encoding and Variation Operators

We used the same binary string encoding as in [57]. A chromosome (individual) is formed by a sequence of test cases, where each test case is represented by l bits (l being the number of features in the feature model). Thus, a test suite with k test cases has $k \times l$ bits. The order of features in each test case corresponds to the depth-first traversal order of the tree. A value of 1 means that the corresponding feature is included in the test case, while a value of 0 means the feature not being included. Note that all of the products (tests) used are valid products as a result of using two tools that generate valid products: CASA [25] in seeding and PLEDGE [30] (where internally a modified version of SAT4j [29] is used) when adding products.

We used four simple mutation operators from [57] to produce new individuals: test case swap, test case removal, test case substitution, and test case addition. Test case swap, which changes the order of test cases, exchanges the order of two randomly chosen test cases in an individual. The three test case selection operators (test case removal, test case substitution and test case addition) remove, substitute and add one test case at a randomly chosen place of the test suite (i.e., the individual), respectively.

3.5 Procedure of the Proposed Algorithm GrES

Algorithm 1 gives the framework of the proposed algorithm GrES. Step 1 is the archive initialisation. Steps 2–9 is the main part of the algorithm. First, the current evolutionary population $\{\mu_1, \mu_2, \mu_3\}$ is selected from the archive set by three selection operations (explained previously in Section 3.3), as stated in Step 3. Then, three variation operations (Algorithms 2 and 3) are carried out on μ_1, μ_2 and μ_3 and obtain three offspring λ_1, λ_2 and λ_3 (Step 4). Step 5 compares these three offspring with the individuals in the archive set. The comparison operation, which is based on the approach in Section 3.1, is described in Algorithm 4. Finally, Steps 6–8 maintain the archive set—if its size exceeds the predefined maximum threshold, then a truncation operation is performed to remove the individual(s) in the current most crowded area. Note that this truncation (described in Algorithm 5) is the same as in PAES [39].

ALGORITHM 2: $VariationSR(\mu)$

1: **if** $random(0, 1) \leq 0.5$ **then**
2: $\lambda \leftarrow TestCase\_Swap(\mu)$
3: **else**
4: $\lambda \leftarrow TestCase\_Removal(\mu)$
5: **end if**
6: **return** λ

ALGORITHM 3: $VariationSRAS(\mu)$

```
prob ← random(0, 1)
if prob ≤ 0.25 then
    λ ← TestCase_Swap(μ)
else if 0.25 < prob ≤ 0.5 then
    λ ← TestCase_Removal(μ)
else if 0.5 < prob ≤ 0.75 then
    λ ← TestCase_Addition(μ)
else
    λ ← TestCase_Substitution(μ)
end if
return λ
```

ALGORITHM 4: $Join(Q, \lambda)$

```
for all q ∈ Q do
    if Coverage(q) > Coverage(λ) then
        return Q
    else if Coverage(q) < Coverage(λ) then
        Q ← Q \ {q}
    else if q ≺ λ then
        return Q
    else if λ ≺ q then
        Q ← Q \ {q}
    end if
end for
Q ← Q ∪ {λ}
return Q
```

ALGORITHM 5: $Truncate(Q, N)$

```
while |Q| > N do
    Q_c ← FindMostCrowded(Q)
    /* Pick out the most crowded hyperbox, in which the solution set are denoted as Q_c */
    q ← RandomPick(Q_c)
    Q ← Q \ {q}
end while
return Q
```

3.6 The Optimisation Problem

In this subsection, we present nine state-of-the-art objective functions for search-based test case generation in SPLs. Three of the functions concern test case selection (i.e., selecting a subset of products to be tested) and six functions address test case prioritisation (i.e., ordering products to detect bugs as soon as possible). All the functions receive an attributed feature model representing the SPL under test (*fm*) and a test suite (*ts*) as inputs and return a value measuring the quality of the suite with respect to the optimisation objective. Each function is illustrated using the feature model *fm* from Figure 1 and the test suite $ts = [TC1, TC2]$ shown in Table 3. Table 4 summarises the

Table 3. Example Test Cases

ID	Test Case
TC1	Mobile Phone, Calls, Screen, Basic, Media, MP3
TC2	Mobile Phone, Calls, Screen, HD, GPS, Media, Camera, MP3

Table 4. Objective Functions

Objective function	Goal	Optimization	Section
Pairwise coverage	Test selection	Maximise	3.6.1
Test suite size	Test selection	Minimise	3.6.2
Test suite cost	Test selection	Minimise	3.6.3
Coefficient of connectivity-density	Test prioritisation	Maximise	3.6.4
Dissimilarity	Test prioritisation	Maximise	3.6.5
Variability coverage and cyclomatic complexity	Test prioritisation	Maximise	3.6.6
Number of changes	Test prioritisation	Maximise	3.6.7
Number of faults	Test prioritisation	Maximise	3.6.8
Feature size	Test prioritisation	Maximise	3.6.9

objective functions used in our study, indicating for each objective the section where it is described and whether it will be maximised or minimised.

We use the following objective functions for test case selection in SPLs.

3.6.1 Pairwise Coverage (PC). This function counts the number of feature pairs covered by the input test suite. It has been extensively used in related work in SPL testing [52]. In this article, we use the most common approach where the function $pc(fm, tc_i)$ returns the number of pairs of features covered by the test case tc_i at position i in test suite ts, that were not covered by preceding test cases $tc_1, \ldots tc_{i-1}$. This objective function is defined as follows:

$$PC(fm, ts) = \sum_{i=1}^{|ts|} pc(fm, tc_i). \tag{1}$$

Test case TC1 covers 36 different pairs of features such as the pair [`Calls`, ¬`GPS`], which indicates that the feature `Calls` is selected in TC1 and the feature `GPS` is not selected. Analogously, test case TC2 covers 27 different pairs of features, e.g., [`HD`, `GPS`]. Therefore, the function is calculated as $PC(fm, ts) = 36 + 27 = 63$.

3.6.2 Test Suite Size (TSS). This function returns the number of products in the input test suite ts [31, 47, 48]. This objective function is defined as follows:

$$TSS(ts) = |ts|. \tag{2}$$

In the example, where $ts = [TC1, TC2]$, $TSS(ts) = 2$.

3.6.3 Test Suite Cost (TSC). This function returns the cost of testing the products in a suite as the sum of the cost of testing each product [31, 71]. The cost of testing a product is equal to the sum of the cost of the features in the product. Recall that the cost attribute represents an estimation of the resources needed to test the feature (e.g., execution time). Let the function $cost(fm, tc_i)$ return the sum of the costs of the features included in the test case tc_i. The test suite cost is defined as

follows:

$$TSC(fm, ts) = \sum_{i=1}^{|ts|} cost(fm, tc_i). \quad (3)$$

As an example, based on the feature attribute values depicted in Table 1, the cost of the features in TC1 is 20 for the feature `Calls`, 5 for `Screen`, 25 for `Basic`, 5 for `Media`, and 10 for `MP3`, adding up 65 in total. Analogously, the sum of the cost of the features of TC2 is 145. Therefore, the value of the function is calculated as $TSC(fm, ts) = 65 + 145 = 210$.

In the following, we present the objective functions for test case prioritisation in SPLs used in our work, proposed by Parejo et al. [57]. We start by introducing the prioritisation objectives based on the information extracted from the feature tree.

3.6.4 Coefficient of Connectivity-Density (CoC). The coefficient of connectivity-density metric calculates the complexity of a feature model in terms of the number of edges and constraints of the model [7]. Sánchez et al. [60] adapted this metric to calculate the complexity of a feature by measuring the number of edges and constraints in which the feature is involved. For a feature f, the complexity of f is the number of edges (including those for cross-tree constraints) that are associated with f in the feature model. For example, `Mobile Phone` has complexity 4 (there are four children of `Mobile Phone` in the feature model) and `Camera` has complexity 2 (one for the edge to its parent and one for the cross-tree constraint).

This function calculates the CoC of the features of a suite, thus giving priority to those test cases covering features with higher CoC more quickly. Let the function $coc(fm, tc_i)$ return the sum of the CoC metric of the features included in the test case tc_i, considering only those features not included in preceding test cases $tc_1 \ldots tc_{i-1}$. This objective function is defined as follows:

$$CoC(fm, ts) = \sum_{i=1}^{|ts|} \frac{coc(fm, tc_i)}{i} \quad (4)$$

As an example, test case TC1 has a CoC of 13 calculated as follows: 4 edges in `Mobile Phone`, 1 edge in `Calls`, 3 edges in `Screen`, 1 edge in `Basic`, 3 edges in `Media` and 1 edge in `MP3`. The features in TC2 that are not included in TC1 are `HD`, `GPS`, and `Camera`. Hence TC2 has a value of 5 computed as follows: 2 edges in `HD`, 1 edge in `GPS`, and 2 edges in `Camera`. Considering that TC1 is placed in position 1 and TC2 in position 2, the function is calculated as $CoC(fm, ts) = (13/1) + (5/2) = 13 + 2.5 = 15.5$

3.6.5 Dissimilarity (D). Some authors have shown that dissimilar products are likely to detect more faults than similar ones [3, 29, 60]. Based on this idea, we let the dissimilarity of a test case be the number of features of this test case that have not been included in the preceding test cases, giving priority to those test cases that cover more new features more quickly. Let the function $df(fm, tc_i)$ return the number of features in test case tc_i that were not included in the preceding test cases $tc_1 \ldots tc_{i-1}$. The dissimilarity objective function is defined as follows:

$$D(fm, ts) = \sum_{i=1}^{|ts|} \frac{df(fm, tc_i)}{i} \quad (5)$$

As an example, test case TC1 has a dissimilarity value of 6, equal to the number of its features. Test case `TC2` has a dissimilarity value of 3, equal to the number of features not included in TC1: `HD`, `GPS` and `Camera`. Considering that TC1 is placed in position 1 and TC2 in position 2, the function is calculated as $D(fm, ts) = (6/1) + (3/2) = 6 + 1.5 = 7.5$.

3.6.6 Variability Coverage and Cyclomatic Complexity (VCCC). The variability coverage and the cyclomatic complexity metrics of a product have been used as effective drivers for the selection [22] and prioritisation [60] of test cases in SPLs. The *cyclomatic complexity* of a product is measured as the number of cross-tree constraints enforced by the product. The *variability coverage* is measured as the number of variation points in the products, where a *variation point* is any feature that provides different variants to create a product, i.e., optional features and non-leaf features with one or more non-mandatory subfeatures. This objective function calculates the value of both metrics for each product of the suite, giving a higher weight to those products at the top of the suite. Let function $vc(fm, tc_i)$ return the number of different cross-tree constraints and the number of variation points involved on the features included in test case tc_i that were not included in preceding test cases $tc_1 \ldots tc_{i-1}$. The VCCC objective function is defined as follows:

$$VCCC(fm, ts) = \sum_{i=1}^{|ts|} \frac{vc(fm, tc_i)}{i} \tag{6}$$

As an example, TC1 has three variation points, namely `Mobile Phone`, `Screen`, and `Media`. Analogously, the features in TC2 not included in TC1 include one variation point (`GPS`) and a cross-tree constraint between `HD` and `Camera`, which counts as 2. Since TC1 is placed in position 1 and TC2 in position 2, the function is calculated as $VCCC(fm, ts) = (3/1) + (3/2) = 3 + 1.5 = 4.5$.

The following objective functions are based on the extra-functional information provided by feature attributes. To illustrate each function, the feature attribute values depicted in Table 1 are used.

3.6.7 Number of Changes (NC). The number of changes has been proposed as a good indicator of error proneness and can be used to predict faults in later versions of systems [77]. This objective function counts the number of changes that have been made to the (code of the) features contained in the products of a test suite, giving more weight to the features included in the products at the top of the test suite. Let the function $nc(fm, tc_i)$ return the number of code changes covered by features of the test case tc_i at position i that were not covered by preceding test cases $tc_1 \ldots tc_{i-1}$. Note that we say that a test case covers a change if it includes the feature where the change was made. The NC objective function is defined as follows:

$$NC(fm, ts) = \sum_{i=1}^{|ts|} \frac{nc(fm, tc_i)}{i} \tag{7}$$

As an example, test case TC1 covers 6 changes in the feature `Calls` (see feature attribute values in Table 1), 2 changes in `Screen`, 1 change in `Basic`, 9 changes in `Media`, and 4 in the feature `MP3`, 22 changes in total. Test case TC2 includes three new features `HD`, `GPS`, and `Camera`, with 3, 8, and 12 changes respectively, 23 changes in total. Considering that TC1 is placed in position 1 and TC2 in position 2, the function is calculated as $NC(fm, ts) = (22/1) + (23/2) = 22 + 11.5 = 33.5$.

3.6.8 Number of Faults (NF). The number of faults in previous versions of the system has also been proposed as an effective bug predictor [77]. This objective function calculates the number of known faults in the features of a test suite and the speed in covering those faults (features), giving a higher value to those test cases that cover more faults faster. We may recall that this objective function uses historical data about the faults reported in previous versions of the products under test. Let function $nf(fm, tc_i)$ return the number of faults detected by the test case tc_i that were not detected by preceding test cases $tc_1 \ldots tc_{i-1}$. Note that we consider that a test case (product) detects a fault if it includes the feature(s) that triggered the fault. The NF objective function is

defined as follows:

$$NF(fm, ts) = \sum_{i=1}^{|ts|} \frac{nf(fm, tc_i)}{i} \tag{8}$$

As shown in Table 1, test case TC1 covers 5 faults in the feature `Calls`, 1 fault in feature `Screen`, 0 faults in feature `Basic`, 5 faults in feature `Media`, and 3 faults in feature `MP3`; 14 faults in total. Test case TC2 covers 2 known faults in `HD`, 11 faults in `GPS`, and 7 faults in `Camera`; 20 faults in total. Assuming that TC1 and TC1 are placed at the first and second positions of the suite respectively, the function is calculated as $NF(fm, ts) = (14/1) + (20/2) = 14 + 10 = 24$.

3.6.9 Feature Size (FS). The size of a feature has also been suggested as a measure that provides a rough idea of the complexity of the feature and its error proneness [59]. This objective function measures the number of lines of code in the features involved in a test suite, giving priority to test suites covering code faster. Let function $fs(fm, tc_i)$ return the sum of the sizes of the features included in the test case tc_i that were not included in preceding test cases $tc_1 \ldots tc_{i-1}$. The FS objective function is defined as follows:

$$FS(fm, ts) = \sum_{i=1}^{|ts|} \frac{fs(fm, tc_i)}{i} \tag{9}$$

Based on the feature attribute values shown in Table 1, test case TC1 has 3,690 lines of code calculated as follows: 1,000 in the feature `Calls`, 930 in `Screen`, 270 in `Basic`, 1,100 in `Media` and 390 in `MP3`. The new features included in TC2 have a total of 1,650 lines of code: 510 in `HD`, 460 in `GPS`, and 680 in `Camera`. Assuming that TC1 is placed in position 1 and TC2 in position 2, the function is calculated as $FS(fm, ts) = (3{,}690/1) + (1{,}650/2) = 3{,}690 + 825 = 4{,}515$.

4 EXPERIMENTAL DESIGN

4.1 Performance Metrics

We used a comprehensive performance metric, hypervolume (HV) [81], to study the results of the experiments. HV measures the volume of the objective space enclosed by a solution set and a reference point. A large value is preferable and reflects the solution set achieving good quality in all three aspects, convergence, extensity and uniformity; an illustration can be seen in [34]. HV is arguably the most popular metric in the area due to its good theoretical and practical properties, such as being compliant with Pareto dominance between solution sets and not requiring a good representation of the problem's Pareto front. There do exist other performance metrics frequently used in software engineering, such as those presented in [72] including GD, IGD, PFS, C, ED, GS, and *epsilon*-indicator. However, each metric has its own working range and applying metrics to problems for which they are not suitable may easily lead to inaccurate evaluation [45]. For example, the metrics GD and IGD need a Pareto front representation with densely and uniformly distributed points [45], which is not available in our problem. The metrics PFS and C consider the number of nondominated solutions in the final population; however, in many-objective optimisation, almost all the solutions in the population are nondominated. The metric ED can only partially reflect convergence [42] and GS only works for bi-objective problems [42]. The metric *epsilon*-indicator tends to give very similar results to HV, as shown theoretically [10] and experimentally [58].

In the calculation of HV, two crucial issues are the scaling of the objective space and the choice of the reference point. Since the objectives in the considered problems take different ranges of values, we normalised the objective values of the obtained solutions according to the ranges of an estimated Pareto front. Following common practice, the estimated Pareto front consists of the nondominated solutions of the collection of all the solutions produced on a given problem. The

reference point was set to 1.1 times the upper bound of the estimated Pareto front (i.e., $r = 1.1^m$) to emphasise the quality balance between convergence and diversity. In addition, since an exact calculation of the HV metric can be computationally expensive in many-objective optimisation, we estimated the HV result by Monte Carlo sampling. Here, 10,000,000 sampling points were used to ensure accuracy [5].

Since the software engineer is typically only interested in test suites that achieve full pairwise coverage, we also introduced a metric to evaluate the ability of each algorithm to return such test suites, called full coverage ratio (FCR), namely, the proportion of solutions with full pairwise coverage in the final population. In addition, it is necessary to note that in computing HV, we only used solutions with full pairwise coverage. That is, we computed HV of only the solutions with full pairwise coverage in the population, considering all objectives except the first one (the pairwise coverage).

4.2 Overview

The proposed approach, GrES, was evaluated through experiments that compared it with several alternatives. The experiments compared FCR and HV values and addressed the following five research questions.

- *Research Question 1 (RQ1)*: How does GrES perform, in terms of FCR and HV, when compared with current solutions to the problem (the two variants of NSGA-II and PLEDGE)?
- *Research Question 2 (RQ2)*: How large are the HV values of the populations returned by the evolution strategy, when compared with the results of alternative representative multi-objective optimisation algorithms using the same approach (first optimise on pairwise coverage)?
- *Research Question 3 (RQ3)*: How large are the HV values of the populations returned by GrES when compared with previous approaches if we restrict the set of objectives to those previously considered?
- *Research Question 4 (RQ4)*: How large are the differences in the HV values of the populations returned in the experiments used for RQ1-3?
- *Research Question 5 (RQ5)*: How long do the different algorithms take?

The initial experiments, addressing the first research question (RQ1) compared GrES with two related approaches proposing the use of different instances of NSGA-II, which we call NSGA-II$_H$ [48] and NSGA-II$_P$ [57]. We chose these approaches because we found them easy to reproduce and because they are both closely related to our work (they addressed the problems of test selection [48] and test prioritisation [57] in SPL).

We also used PLEDGE (Product Line Editor and Test Generation Tool) [30] in the experiments that addressed the first research question. PLEDGE is an open-source tool for the selection and prioritisation of SPL products maximising the feature interactions covered. The key feature of PLEDGE is that it maximises the dissimilarity among products as a proxy to achieve high coverage, allowing it to scale to large SPLs. More specifically, the tool works in two steps. First, a set of products maximising dissimilarity is selected. Second, the products are ordered (i.e., prioritised) based on dissimilarity, i.e., the most different products are ranked first. In practice, PLEDGE aims to optimise two of the objectives presented in Section 3.6: pairwise coverage (selection) and dissimilarity (prioritisation).

PLEDGE receives as inputs a feature model, the number of products to be generated, and the time allowed for generating them. In our experiment, we set the number of products equal to the size of the pairwise suite generated by CASA, used as seed in GrES (between 6 and 61 products). For the execution time limit, we used the median of the execution times of all the other techniques

under evaluation for a model (reported in the next sections). Note that, when comparing PLEDGE and GrES, GrES has the advantage that it starts with a seed that provides full pairwise coverage. GrES, however, is required to optimise over nine objectives whereas PLEDGE only has to optimise on two (a multi-objective problem).

We then performed additional experiments that addressed the second research question (RQ2) and so compared GrES with alternative multi-objective optimisation algorithms when using the same approach (i.e., by first considering pairwise coverage). The aim of these experiments was to evaluate our novel evolution strategy (i.e., if RQ1 found that GrES outperformed the current techniques, then we wanted to know whether this was entirely due to it first optimising on pairwise coverage). These alternative evolutionary algorithms were NSGA-II [17], IBEA [80], MOEA/D [78], SPEA2+SDE [44], and PAES [39]. The first three are representative approaches of Pareto-based, indicator-based, and decomposition-based genetic algorithms, respectively. The main difference among them lies in the environmental selection operation (i.e., selection for survival), which uses the Pareto dominance, performance indicator, and decomposition (by a set of weight vectors), respectively. SPEA2+SDE is a genetic algorithm designed for many-objective optimisation. Very recently, it has been found to perform in general best out of various state-of-the-art many-objective algorithms in several experimental studies [41, 43]. PAES is the most popular evolution strategy which has a (1+1) evolution form. It shares the grid-based environmental selection with our algorithm. In these experiments, all approaches returned populations will 100% FCR and so we only compared the HV values.

The parameter settings used in the experiments will be discussed along with implementation details in Section 4.4. In order to address RQ3, we compared the HV values of populations returned by GrES with those produced by previous approaches that only aim to optimise a subset of the objectives. We are not only interested in the relative HV values but also whether the differences are substantial and so we computed effect sizes (RQ4). Finally, we recorded execution times for a range of experiments (RQ5).

In the rest of this section, we outline the experimental setup; in the next section, we explain what experiments were performed and give the results of these.

4.3 Experimental Subjects

For the evaluation of the approach, we used a set of 20 realistic feature models whose characteristics are depicted in Table 5. By *realistic,* we mean that all the models are related to actual SPLs whose code is publicly available or accessible under request to the respective authors [51, 59]. For each model, the number of features, number of CTCs, number of products and application domain are presented. These models have been used as benchmarks to assess related SPL testing techniques [47, 48, 51, 57, 59].

Experiments were also performed on three groups of 10 randomly-generated feature models, with 30, 50, and 100 features, respectively. These models were generated using the tool BeTTy [63] with the following parameters: 25% mandatory features; 25% optional features; 25% disjunctive relationships; 25% alternative relationships; maximum branching factor of 12; maximum of 5 subfeatures; and 10% CTC (with respect to the number of features). These values have been found to be common in feature models [66]. All models are valid (i.e., they represent at least one valid product) and have no dead features, i.e., all features are part of at least one valid product. Additional feature models, with 500 features, were also used to address the last research question (regarding scalability).

For the calculation of the objective functions, the subject models were extended with four attributes with randomly generated values: cost, number of faults, number of changes, and code size. This is a common approach to evaluate optimisation approaches on attributed feature models

Table 5. Realistic Feature Models

Feature model	Features	CTC	Products	Domain
Apache	10	0	256	Web server
argo-uml-spl	11	0	192	UML tool
BerkeleyDatabase (BDB)	117	282	32	Database
BDBFootprint	9	0	256	Database
BDBMemory	19	0	3,840	Database
BDBPerformance	27	0	1,440	Database
Curl	14	0	1,024	Data transfer
DesktopSearcher	22	0	462	File search
Drupal	48	21	$2.09 \cdot 10^9$	Web framework
fame_dbms_fm	20	0	320	Database
gpl	18	13	73	Graph algorithms
LinkedList	27	0	1,344	Data structures
LLVM	12	0	1,024	Compiler
PKJab	12	0	72	Messenger
Prevayler	6	0	32	Object persistence
SensorNetwork	27	7	16,704	Networking
TankWar	37	0	1,741,824	Game
Wget	17	0	8,192	File retrieval
x264	17	0	2,048	Video encoding
ZipMe	8	0	64	Data compression

[27, 56, 61]. The cost attribute was assigned real values between 0 and 10. The rest of the attributes were assigned integer values between 0 and 10. Additionally, for each model, we generated a matrix of feature pairs representing integration faults. In particular, for each valid feature pair (calculated using the tool SPLCAT [36]), we randomly simulated either 1 or 2 faults with a probability of 0.03. This probability is based on the percentage of integration faults observed in a recent case study [59]. Exceptionally, the Drupal feature model is provided with real values for the aforementioned attributes (except cost) and integration faults extracted from the Drupal Git repository and the Drupal issue tracking system [59]. In this subject, we used the real values for all the attributes except the cost, for which random values were generated as previously described.

4.4 Implementation Details

All the peer algorithms were executed 30 times for each experiment to reduce the impact of their stochastic nature. The termination criterion used in all the algorithms was a predefined number of evaluations, which was set to 10,000. The size of the population for the genetic algorithms except MOEA/D was set to 100. The population size in MOEA/D, which is the same as the number of weight vectors, cannot be specified arbitrarily, so we followed the common practice of using the closest integer to 100 amongst the possible values. The evolution form of PAES was $(\mu+\lambda)$ with $\mu = \lambda = 1$, and the archive size was set to 100. Some of the algorithms require several parameters to be set. As suggested in their original papers, the neighbourhood size was set to 10% of the population size in MOEA/D and the scaling factor κ to 0.05 in IBEA. The number of grid divisions in PAES and GrES was set to 10. In addition, it is worth noting that the cell-centred calculation in the original PAES makes it unable to work in many-objective optimisation [14]. To address this issue, we modified the implementation of PAES by considering the individual-centred calculation [76],

as the same used in GrES. All the algorithms under evaluation searched over valid products only, i.e., all the test cases meet the constraints defined in the feature model.

The single point crossover and four mutation operators from [57] (test case swap, test case removal, test case substitution and test case addition) were used to produce offspring in all the genetic algorithms. To mutate a solution, the four operators were randomly selected and the mutation probability was $1/n$, where n denotes the number of the solution's decision variables (i.e., its suite size). As for crossover probability, 0.8 was used according to the practice in [48]. As a result of using standard values from the literature, we did not require a tuning phase.

All algorithms except one were implemented using C++. The one exception, NSGA-II$_P$, was coded in Java. C++ programs were executed in a notebook computer equipped with an Intel(R) Core(TM)i5-5200 U CPU@2.20 GHz and 8 GB of RAM running Window 7 Ultimate. The Java program was executed in a desktop computer equipped with Intel(R) Core(TM)i7-4770 CPU@3.40 GHz and 16 GB of RAM, running Windows 10 Education.

5 RESULTS

In this section, we outline the experiments carried out to evaluate GrES and the results of these experiments. The experiments used the settings described in the previous section. The HV results in all tables give the mean and standard deviation over 30 independent runs, and the best/better mean for each problem instance is highlighted in boldface. Moreover, in order to have statistically sound conclusions, we adopted the Wilcoxon's rank sum test at a 0.05 significance level to examine the significance of the difference between the results obtained by GrES and its competitors. Symbol "†" indicates that the difference is statistically significant.

5.1 Research Question 1: Comparison with Existing Approaches

The main purpose of this comparison was to verify the proposed optimisation framework in Section 3 (i.e., first consider pairwise coverage and then the other objectives). In order to do this, we compared the proposed (GrES) approach with the two previous approaches (NSGA-II$_H$ and NSGA-II$_P$) and the well-established tool (PLEDGE) as a baseline algorithm. As previously explained, we used two metrics: full coverage ratio FCS (the proportion of solutions in the final population that had full pairwise coverage) and the hypervolume (HV) of these solutions. No results are given for NSGA-II$_P$ for models with 100 features since the computation time was excessive. Note that since on most of the models there is no solution obtained by PLEDGE reaching the full pairwise coverage (consequently, both FCS and HV being zero), for comparing GrES with PLEDGE, we show the objective values of the best solution in terms of pairwise and dissimilarity.

Consider first the comparison with NSGA-II$_H$ and NSGA-II$_P$. The results of the experiments with randomly generated feature models can be found in Tables 6, 7, and 8. The first observation is that GrES always returned a population in which all test suites provided 100% pairwise coverage. This is in contrast with the other two approaches, which in all cases had a mean FCR of less than 10%. Thus, the GrES approach provides the Software Engineer with more test suites that are suitable (that provide full pairwise coverage).

If we consider the HV values, then we can see that the proposed approach is best in the majority of experiments with models that had 30 features and was best in all experiments with larger feature models. The differences were found to be statistically significant. Thus, not only does GrES provide more suitable solutions (test suites), it also provides a relatively diverse set of solutions. As a result, the Software Engineer has more test suites to choose from and also a wider range of trade-offs between objectives.

The potential weakness of using randomly generated experimental subjects is that they might not be representative of real examples (feature models). It is therefore promising that similar results

Table 6. FCR and HV of NSGA-II$_H$, NSGA-II$_P$ and GrES on the 10 Random Models with 30 Features

Problem	NSGA-II$_H$		NSGA-II$_P$		GrES	
	FCR	HV	FCR	HV	FCR	HV
Model30-1	7.03%	1.371E-03(8.7E-04)†	1.80%	**4.121E-02(1.0E-01)**†	100%	2.218E-02(3.2E-02)
Model30-2	7.27%	5.315E-03(2.1E-02)†	2.17%	3.612E-02(6.0E-02)†	100%	**1.875E-01(7.4E-02)**
Model30-3	7.13%	2.342E-04(3.5E-04)†	1.03%	2.707E-03(1.4E-02)†	100%	**3.943E-02(1.8E-02)**
Model30-4	7.07%	1.115E-04(1.9E-04)†	2.93%	**4.694E-02(1.0E-01)**†	100%	3.466E-02(1.0E-02)
Model30-5	6.80%	7.145E-07(1.4E-06)†	1.07%	**4.264E-02(2.3E-01)**†	100%	2.783E-03(2.7E-03)
Model30-6	6.70%	4.502E-06(1.7E-05)†	1.13%	6.076E-02(2.9E-01)†	100%	**2.057E-01(4.4E-02)**
Model30-7	6.13%	3.838E-04(2.1E-03)†	1.07%	1.143E-06(6.3E-06)†	100%	**5.731E-01(2.9E-01)**
Model30-8	7.10%	2.844E-05(1.6E-04)†	1.27%	1.066E-03(5.8E-03)†	100%	**4.512E-02(5.8E-02)**
Model30-9	6.30%	4.813E-02(1.6E-03)†	1.17%	2.648E-02(1.2E-02)†	100%	**3.230E-01(1.2E-01)**
Model30-10	7.70%	0.000E+00(0.0E+00)†	1.13%	5.536E-04(2.6E-03)†	100%	**1.970E-01(8.5E-02)**

Table 7. FCR and HV of NSGA-II$_H$, NSGA-II$_P$ and GrES on the 10 Random Models with 50 Features

Problem	NSGA-II$_H$		NSGA-II$_P$		GrES	
	FCR	HV	FCR	HV	FCR	HV
Model50-1	8.93%	1.467E-03(8.0E-03)†	1.03%	2.758E-02(3.9E-02)†	100%	**2.481E-01(6.7E-02)**
Model50-2	5.77%	0.000E+00(0.0E+00)†	1.03%	0.000E+00(0.0E+00)†	100%	**5.370E-01(2.2E-01)**
Model50-3	5.63%	0.000E+00(0.0E+00)†	1.07%	1.072E-05(5.6E-05)†	100%	**2.707E-01(2.1E-01)**
Model50-4	5.77%	0.000E+00(0.0E+00)†	1.07%	8.574E-07(2.7E-06)†	100%	**9.165E-01(7.6E-02)**
Model50-5	7.17%	1.055E-04(5.8E-04)†	1.03%	2.065E-05(9.4E-05)†	100%	**5.901E-01(1.9E-01)**
Model50-6	5.20%	0.000E+00(0.0E+00)†	1.10%	6.552E-03(3.6E-02)†	100%	**1.517E-02(3.0E-03)**
Model50-7	5.37%	0.000E+00(0.0E+00)†	1.00%	0.000E+00(0.0E+00)†	100%	**4.768E-01(6.9E-02)**
Model50-8	6.20%	3.676E-02(2.5E-02)†	1.03%	3.688E-02(1.6E-02)†	100%	**5.056E-01(5.1E-02)**
Model50-9	6.87%	2.144E-07(1.2E-06)†	1.03%	6.236E-03(1.5E-02)†	100%	**4.272E-01(1.6E-01)**
Model50-10	6.23%	0.000E+00(0.0E+00)†	1.07%	1.168E-02(1.7E-02)†	100%	**4.933E-01(1.3E-01)**

Table 8. FCR and HV of NSGA-II$_H$ and GrES on the 10 Random Models with 100 Features

Problem	NSGA-II$_H$		GrES	
	FCR	HV	FCR	HV
Model100-1	2.27%	1.572E-06(8.6E-06)†	100%	**5.076E-01(1.9E-01)**
Model100-2	2.67%	0.000E+00(0.0E+00)†	100%	**6.737E-01(1.2E-01)**
Model100-3	2.80%	5.773E-03(3.2E-02)†	100%	**9.572E-01(6.8E-02)**
Model100-4	3.33%	0.000E+00(0.0E+00)†	100%	**8.153E-01(8.4E-02)**
Model100-5	4.17%	7.860E-07(4.3E-06)†	100%	**2.087E-01(8.1E-02)**
Model100-6	4.07%	0.000E+00(0.0E+00)†	100%	**2.519E-01(1.8E-01)**
Model100-7	4.90%	0.000E+00(0.0E+00)†	100%	**6.107E-01(3.7E-02)**
Model100-8	1.90%	1.149E-03(1.3E-04)†	100%	**1.199E+00(1.7E-01)**
Model100-9	3.60%	3.573E-07(9.9E-07)†	100%	**8.971E-01(8.2E-02)**
Model100-10	3.67%	0.000E+00(0.0E+00)†	100%	**1.802E-01(4.9E-02)**

Table 9. FCR and HV of NSGA-II$_H$, NSGA-II$_P$, and GrES on the 19 Realistic Models

Problem	NSGA-II$_H$		NSGA-II$_P$		GrES	
	FCR	HV	FCR	HV	FCR	HV
Apache	5.53%	0.000E+00(0.0E+00)†	1.10%	0.000E+00(0.0E+00)†	100%	**3.417E-02(7.3E-02)**
argo-uml-spl	6.00%	4.746E-04(5.9E-04)†	2.53%	**4.657E-03(1.0E-02)**†	100%	2.322E-05(7.6E-06)
BerkeleyDB	7.83%	1.915E-05(3.4E-05)†	5.27%	**1.004E-02(3.8E-02)**†	100%	7.753E-05(7.2E-05)
BerkeleyDBFootprint	6.47%	1.273E-04(2.8E-05)†	1.77%	1.054E-03(1.6E-03)†	100%	**9.718E-03(1.1E-02)**
BerkeleyDBMemory	3.47%	0.000E+00(0.0E+00)†	1.03%	1.864E-04(1.0E-03)†	100%	**1.232E-01(1.0E-01)**
BerkeleyDBPerformance	6.30%	1.118E-04(4.0E-04)†	2.30%	1.608E-02(3.7E-02)†	100%	**1.522E-01(5.2E-02)**
Curl	9.40%	0.000E+00(0.0E+00)†	3.57%	3.614E-02(1.2E-01)†	100%	**1.263E-01(4.4E-02)**
DesktopSearcher	5.87%	7.159E-03(1.5E-02)†	3.73%	6.168E-03(7.8E-03)†	100%	**1.196E-01(6.0E-02)**
fame-dbms-fm	6.73%	2.144E-07(6.5E-07)†	1.27%	1.859E-02(4.6E-02)†	100%	**4.048E-01(2.9E-02)**
gpl	7.47%	0.000E+00(0.0E+00)†	2.90%	**4.254E-02(1.9E-01)**	100%	4.269E-04(1.0E-04)
LinkedList	6.20%	1.140E-03(3.8E-03)†	1.70%	2.732E-03(4.2E-03)†	100%	**2.950E-02(1.4E-02)**
LLVM	6.10%	1.225E-04(1.6E-05)†	3.60%	1.766E-03(2.7E-03)†	100%	**1.234E-01(1.6E-01)**
PKJab	6.57%	5.923E-05(2.5E-04)†	4.13%	5.785E-04(4.6E-04)†	100%	**5.550E-02(1.1E-01)**
Prevayler	7.60%	5.922E-04(3.2E-03)†	0.87%	1.899E-03(8.3E-03)†	100%	**5.039E-02(5.9E-02)**
SensorNetwork	6.67%	1.058E-05(5.8E-05)†	1.40%	2.843E-03(8.0E-03)†	100%	**1.009E-01(7.3E-02)**
TankWar	6.73%	0.000E+00(0.0E+00)†	1.00%	1.770E-03(6.7E-03)†	100%	**3.395E-01(1.6E-01)**
Wget	7.60%	0.000E+00(0.0E+00)†	1.73%	1.663E-03(3.7E-03)†	100%	**1.968E-02(2.1E-02)**
x264	5.33%	0.000E+00(0.0E+00)†	1.87%	2.237E-02(4.5E-02)†	100%	**5.057E-02(5.4E-03)**
ZipMe	6.40%	8.860E-06(3.9E-05)†	1.33%	0.000E+00(0.0E+00)†	100%	**4.124E-02(5.3E-02)**

Table 10. FCR and HV of NSGA-II$_H$, NSGA-II$_P$ and GrES on the Real Model DrupalFM

Problem	NSGA-II$_H$		NSGA-II$_P$		GrES	
	FCR	HV	FCR	HV	FCR	HV
DrupalFM	8.93%	5.097E-03(1.6E-03)†	1.00%	6.722E-03(8.2E-03)†	100%	**6.789E-01(2.2E-01)**

were found for the realistic feature models (Table 9). Specifically, in all cases GrES had an FCR of 100%, while the other approaches always had an FCR of less than 10%. In addition, in 16 of the 19 cases, the HV of GrES was highest and the differences were found to be statistically significant. Overall, it is clear that GrES outperformed the other two approaches in terms of both FCR and HV. Table 10 gives the results for the Drupal feature model with real attribute values, with GrES again being most effective.

Regarding the comparison between PLEDGE and GrES, since on most of the models there is no solution returned by PLEDGE having full pairwise coverage (thus both FCS and HV were zero), we do not show the FCS and HV results here. Instead, we show the objective values of the best solution in terms of the pairwise coverage and the dissimilarity which PLEDGE aims to optimise, on the random models and the realistic/real models in Table 11 and Table 12, respectively. For PLEDGE, the best solution is determined first by coverage and then by dissimilarity; for GrES, the best solution is by dissimilarity since all solutions produced have the same coverage. As can be seen in the tables, indeed the pairwise coverage of the solution of PLEDGE on all of the random models is close to but not up to the full value, with some exceptions on realistic models. But we need to note that a seed with full coverage was used in GrES's search. For the objective dissimilarity, GrES always has a better result; similar patterns can be seen in the other five prioritisation objectives.

Table 11. Objective Values of the Best Solution (in Terms of Pairwise Coverage and Dissimilarity) Obtained by PLEDGE and GrES on the 30 Random Models

Problem	Algorithm	PC	NF	FS	NC	VCCC	CoC	D	TSC	TSS
Model30-1	PLEDGE	1226.0	131.3	102.3	87.2	7.2	51.7	23.5	1490.1	16.0
	GrES	**1230.0**	**155.3**	**125.0**	**106.5**	**9.0**	**61.5**	**28.5**	**1234.4**	**12.0**
Model30-2	PLEDGE	1031.0	125.1	111.0	131.9	6.6	46.3	20.3	1238.1	15.0
	GrES	**1033.0**	**177.2**	**148.7**	**175.3**	**9.8**	**60.3**	**27.6**	**860.5**	**10.0**
Model30-3	PLEDGE	1182.0	121.3	137.5	114.8	8.7	54.4	23.8	1366.4	14.0
	GrES	**1191.0**	**139.6**	**145.7**	**141.2**	**10.3**	**58.5**	**26.7**	**1265.4**	**12.0**
Model30-4	PLEDGE	764.0	149.7	146.4	105.3	**7.6**	54.5	22.1	1164.7	16.0
	GrES	**767.0**	**160.8**	**153.8**	**118.3**	7.4	**57.0**	**24.8**	**784.4**	**10.0**
Model30-5	PLEDGE	1391.0	138.6	105.0	92.0	8.2	47.1	20.6	870.9	13.0
	GrES	**1393.0**	**186.1**	**140.8**	**115.0**	**9.9**	**59.2**	**26.7**	**843.1**	**12.0**
Model30-6	PLEDGE	1180.0	138.6	123.8	108.6	9.2	58.4	25.5	1440.2	14.0
	GrES	**1184.0**	**141.6**	**126.1**	**113.9**	**9.9**	**59.6**	**26.1**	**1259.4**	**12.0**
Model30-7	PLEDGE	1223.0	99.9	102.0	97.0	7.0	45.6	19.6	1001.6	17.0
	GrES	**1232.0**	**122.8**	**128.2**	**115.5**	**9.5**	**59.3**	**25.8**	**870.4**	**15.0**
Model30-8	PLEDGE	1266.0	115.0	119.8	108.0	10.2	53.8	23.8	1172.7	13.0
	GrES	**1269.0**	**139.8**	**138.8**	**125.0**	**12.4**	**62.3**	**28.3**	**1016.6**	**11.0**
Model30-9	PLEDGE	1408.0	116.3	106.3	152.4	11.5	54.4	23.9	1288.3	14.0
	GrES	**1416.0**	**139.0**	**125.0**	**171.3**	**12.5**	**61.3**	**27.8**	**1123.2**	**12.0**
Model30-10	PLEDGE	1434.0	101.6	90.1	95.6	7.4	42.8	18.8	1229.4	16.0
	GrES	**1439.0**	**156.5**	**138.5**	**134.3**	**10.7**	**59.8**	**26.8**	**1076.8**	**14.0**
Model50-1	PLEDGE	3030.0	238.6	204.5	190.5	13.7	95.2	40.3	2116.4	11.0
	GrES	**3041.0**	**284.8**	**235.0**	**210.5**	**16.3**	**105.0**	**47.5**	**1958.6**	**10.0**
Model50-2	PLEDGE	4299.0	208.2	148.3	166.7	12.2	79.6	34.1	**2284.6**	17.0
	GrES	**4321.0**	**292.2**	**195.1**	**220.5**	**15.7**	**99.3**	**43.4**	2324.9	17.0
Model50-3	PLEDGE	4204.0	174.2	150.3	143.7	10.9	71.9	31.4	2689.0	22.0
	GrES	**4237.0**	**255.5**	**205.4**	**213.7**	**15.3**	**100.6**	**44.1**	**2514.1**	**20.0**
Model50-4	PLEDGE	4352.0	169.5	149.7	167.6	12.3	73.7	31.9	**2016.9**	**18.0**
	GrES	**4436.0**	**226.1**	**201.1**	**224.9**	**13.3**	**91.3**	**41.1**	2067.5	19.0
Model50-5	PLEDGE	3791.0	129.8	137.2	133.8	9.3	58.9	26.0	**2094.7**	17.0
	GrES	**3819.0**	**206.1**	**240.3**	**217.4**	**15.3**	**100.3**	**43.9**	2422.4	**16.0**
Model50-6	PLEDGE	3824.0	145.9	143.3	189.2	12.8	77.6	32.8	**2593.7**	**21.0**
	GrES	**3842.0**	**203.1**	**203.7**	**250.3**	**18.0**	**101.3**	**44.3**	2729.1	22.0
Model50-7	PLEDGE	3929.0	211.6	196.4	200.9	12.6	88.2	38.5	2010.1	22.0
	GrES	**3974.0**	**211.9**	**202.7**	**210.4**	**16.5**	**94.0**	**40.5**	**1704.0**	**20.0**
Model50-8	PLEDGE	3656.0	186.1	208.7	190.3	**15.0**	97.0	40.6	2696.4	17.0
	GrES	**3684.0**	**204.6**	**211.1**	**208.3**	14.3	**99.1**	**43.5**	**2610.1**	**16.0**
Model50-9	PLEDGE	3964.0	171.8	157.9	140.2	11.8	74.1	31.5	**1862.2**	15.0
	GrES	**3976.0**	**256.6**	**228.2**	**204.2**	**15.9**	**101.2**	**44.7**	1984.3	15.0
Model50-10	PLEDGE	3667.0	129.9	122.0	150.2	8.9	64.8	28.4	**1615.0**	13.0
	GrES	**3706.0**	**219.3**	**192.6**	**221.2**	**14.5**	**100.9**	**44.4**	1823.5	13.0
Model100-1	PLEDGE	18187.0	548.6	374.5	345.7	27.1	175.2	72.8	11001.3	52.0
	GrES	**18429.0**	**632.4**	**392.8**	**376.4**	**28.6**	**182.0**	**76.8**	**9013.2**	52.0

(Continued)

Table 11. Continued

Problem	Algorithm	PC	NF	FS	NC	VCCC	CoC	D	TSC	TSS
Model100-2	PLEDGE	18211.0	442.5	302.9	226.2	19.9	123.4	52.0	9290.8	41.0
	GrES	**18298.0**	**788.4**	**391.4**	**367.7**	**30.7**	**196.9**	**82.5**	**8380.6**	**39.0**
Model100-3	PLEDGE	15483.0	356.4	214.6	225.7	17.4	102.0	44.3	9564.6	42.0
	GrES	**15626.0**	**747.3**	**416.2**	**444.8**	**32.2**	**200.0**	**86.0**	**7891.1**	**36.0**
Model100-4	PLEDGE	18114.0	**772.7**	364.0	395.7	26.1	191.0	81.2	8158.5	33.0
	GrES	**18256.0**	734.3	**445.5**	**420.7**	**26.8**	**196.7**	**86.4**	**7532.8**	**31.0**
Model100-5	PLEDGE	17516.0	529.3	312.3	314.6	21.6	153.1	65.9	6920.4	28.0
	GrES	**17559.0**	**717.2**	**419.7**	**400.3**	**27.5**	**193.2**	**82.6**	**6173.4**	28.0
Model100-6	PLEDGE	16323.0	391.0	223.8	228.2	18.2	106.4	46.2	**7499.7**	31.0
	GrES	**16433.0**	**718.6**	**406.1**	**430.4**	**31.4**	**198.3**	**84.9**	8355.8	**29.0**
Model100-7	PLEDGE	18090.0	530.2	371.2	316.2	22.9	160.9	66.9	**5823.5**	**22.0**
	GrES	**18204.0**	**737.0**	**441.3**	**375.6**	**26.2**	**195.4**	**84.5**	6596.2	24.0
Model100-8	PLEDGE	17511.0	429.1	289.7	263.2	22.1	134.7	54.9	10982.6	63.0
	GrES	**17694.0**	**552.8**	**365.9**	**343.8**	**23.8**	**166.2**	**69.0**	**9578.4**	**58.0**
Model100-9	PLEDGE	17474.0	401.6	243.0	228.1	16.1	108.9	47.7	**7698.0**	**35.0**
	GrES	**17560.0**	**733.3**	**416.1**	**396.4**	**27.8**	**194.2**	**82.7**	8760.8	38.0
Model100-10	PLEDGE	17836.0	544.6	321.0	305.8	26.9	159.5	66.4	**6821.9**	**31.0**
	GrES	**18041.0**	**712.2**	**391.1**	**375.7**	**30.3**	**186.7**	**80.9**	7329.1	33.0

For PLEDGE, the best solution is determined first by coverage and then by dissimilarity; for GrES, the best solution is by dissimilarity as all solutions have the same coverage. A better value is highlighted in boldface. PC: Pairwise coverage, NF: Number of faults, FS: Feature size, NC: Number of changes, VCCC: Variability coverage and cyclomatic complexity, CoC: Coefficient of connectivity-density, D: Dissimilarity, TSC: Test suite cost, TSS: Test suite size.

One interesting exception is on the model BerkeleyDB where the two algorithms have the same values on all the prioritisation objectives. As for the two cost objectives, the solution of PLEDGE does perform better on some models, 12 and 5, respectively, on the test suite cost and test suite size out of all the 50 test models. Nevertheless, it is worth mentioning that even for the models where the solution of PLEDGE in the tables is better in terms of the costs, there may exist other solutions obtained by GrES that were better than PLEDGE's solution for all nine objectives (they were not shown in the tables since they are not the best on dissimilarity).

5.2 Research Question 2: Comparison with Representative Multi-Objective Evolutionary Approaches

The results in the previous subsection did not specifically evaluate the evolution strategy used in GrES. As a result, we also carried out experiments that compared two evolution strategies (GrES and PAES [39]) with four representative genetic algorithms (NSGA-II [17], IBEA [80], MOEA/D [78], and SPEA2+SDE [44]). In order to focus the evaluation on the use of an evolution strategy, all techniques used our optimisation framework in which one first compares solutions based on pairwise coverage and only then, if they have the same pairwise coverage, use the other objectives. The optimisation framework led to an FCR of 100% in all cases and so we only report the HV values.

Tables 13, 14, and 15 give the results with randomly generated models. In all cases, the best technique was one of those that used an evolution strategy (PAES or GrES) and the difference between the best of PAES/GrES and the other multi-objective optimisation algorithms was statistically significant. Interestingly, GrES tended to produce higher HV values than PAES: this was the

Table 12. Objective Values of the Best Solution (in Terms of Pairwise Coverage and Dissimilarity) Obtained by PLEDGE and GrES on the 20 Realistic and Real Models

Problem	Algorithms	PC	NF	FS	NC	VCCC	CoC	D	TSC	TSS
Apache	PLEDGE	145.0	51.3	32.0	40.5	1.0	16.0	8.0	177.6	8.0
	GrES	145.0	**61.0**	**40.0**	**53.0**	1.0	**18.0**	**10.0**	**133.2**	**6.0**
argo-uml-spl	PLEDGE	161.0	39.0	46.0	52.5	3.0	18.5	9.5	379.9	8.0
	GrES	**162.0**	**48.0**	**53.0**	**60.0**	3.0	**20.0**	**11.0**	**339.5**	**7.0**
BerkeleyDB	PLEDGE	14138.0	722.0	539.0	554.0	568.0	778.0	111.0	2900.1	8.0
	GrES	14138.0	722.0	539.0	554.0	568.0	778.0	111.0	**2169.9**	**6.0**
BerkeleyDBFootprint	PLEDGE	128.0	33.3	31.0	42.0	1.0	13.8	6.8	167.1	8.0
	GrES	128.0	**38.0**	**41.0**	**58.0**	1.0	**16.0**	**9.0**	**125.3**	**6.0**
BerkeleyDBMemory	PLEDGE	618.0	71.5	45.9	58.5	3.0	28.9	11.9	1216.0	32.0
	GrES	**623.0**	**82.2**	**59.0**	**71.7**	3.0	**31.4**	**14.4**	**1144.3**	**30.0**
BerkeleyDBPerformance	PLEDGE	903.0	121.0	117.0	96.3	8.0	40.5	20.0	**834.7**	11.0
	GrES	**908.0**	**148.2**	**142.7**	**126.3**	**10.0**	**49.8**	**24.8**	851.7	**10.0**
Curl	PLEDGE	288.0	58.6	62.6	45.3	1.5	21.1	10.6	478.0	10.0
	GrES	**292.0**	**70.7**	**80.3**	**56.5**	**2.0**	**24.8**	**12.8**	**438.6**	**9.0**
DesktopSearcher	PLEDGE	573.0	62.2	74.3	64.8	6.0	29.5	14.5	733.3	11.0
	GrES	**574.0**	**96.8**	**113.5**	**89.2**	**8.0**	**39.8**	**19.8**	**517.6**	**7.0**
fame-dbms-fm	PLEDGE	544.0	88.3	**83.7**	72.6	6.5	32.3	15.3	841.6	12.0
	GrES	**554.0**	**90.4**	81.4	**77.0**	6.5	**32.8**	**15.8**	**697.7**	**10.0**
gpl	PLEDGE	411.0	86.9	84.8	83.5	22.5	50.3	15.1	900.6	14.0
	GrES	411.0	**89.5**	**86.0**	**84.7**	**23.2**	**51.3**	**15.3**	**642.9**	**10.0**
LinkedList	PLEDGE	780.0	94.5	144.3	114.1	9.0	48.2	23.2	1015.5	14.0
	GrES	**793.0**	**107.8**	**161.3**	**123.3**	9.0	**49.6**	**24.6**	**900.2**	**12.0**
LLVM	PLEDGE	219.0	49.8	61.2	52.5	1.0	19.8	9.8	276.6	8.0
	GrES	**220.0**	**59.0**	**71.0**	**65.0**	1.0	**22.0**	**12.0**	**274.8**	**7.0**
PKJab	PLEDGE	177.0	55.9	51.0	59.0	3.0	21.0	11.0	380.1	9.0
	GrES	177.0	**64.0**	**53.0**	**66.0**	3.0	**22.0**	**12.0**	**255.0**	**6.0**
Prevayler	PLEDGE	50.0	34.5	35.0	26.0	1.0	9.5	5.5	175.4	10.0
	GrES	50.0	**39.0**	35.0	**30.0**	1.0	**10.0**	**6.0**	**100.0**	**6.0**
SensorNetwork	PLEDGE	1211.0	83.4	88.3	86.7	13.1	44.8	17.7	1142.3	14.0
	GrES	**1213.0**	**135.0**	**126.5**	**134.5**	**21.5**	**65.0**	**26.0**	**854.2**	**11.0**
TankWar	PLEDGE	2150.0	176.6	152.8	131.2	10.0	60.6	28.1	2089.1	16.0
	GrES	**2157.0**	**200.6**	**180.7**	**162.0**	**11.0**	**67.8**	**32.8**	**1953.2**	**15.0**
Wget	PLEDGE	473.0	58.2	57.0	69.8	2.0	28.3	13.3	645.4	14.0
	GrES	**475.0**	**70.5**	**70.5**	**88.0**	2.0	**30.8**	**15.8**	**576.4**	**13.0**
x264	PLEDGE	465.0	67.8	69.4	56.3	3.0	28.0	13.0	763.7	19.0
	GrES	**469.0**	**77.8**	**76.5**	**63.7**	3.0	**29.7**	**14.7**	**572.8**	**16.0**
ZipMe	PLEDGE	85.0	34.5	39.5	42.5	1.0	13.5	7.5	256.3	9.0
	GrES	85.0	**39.0**	**41.0**	**45.0**	1.0	**14.0**	**8.0**	**152.9**	**6.0**
DrupalFM	PLEDGE	3726.0	2545.3	254567.3	348.3	34.4	105.8	33.8	**1652.6**	13.0
	GrES	**3748.0**	**3326.5**	**325334.0**	**450.0**	**50.1**	**135.0**	**45.0**	1693.8	13.0

For PLEDGE, the best solution is determined first by coverage and then by dissimilarity; for GrES, the best solution is by dissimilarity as all solutions have the same coverage. A better value is highlighted in boldface. PC: Pairwise coverage, NF: Number of faults, FS: Feature size, NC: Number of changes, VCCC: Variability coverage and cyclomatic complexity, CoC: Coefficient of connectivity-density, D: Dissimilarity, TSC: Test suite cost, TSS: Test suite size.

Table 13. HV of the Six Algorithms on the 10 Random Models with 30 Features

Problem	NSGA-II	IBEA	MOEAD	SPEA2+SDE	PAES	GrES
Model30-1	9.182E-03(1.9E-03)†	9.157E-03(1.9E-03)†	6.208E-03(2.9E-03)†	8.888E-03(1.7E-03)†	1.625E-02(2.4E-02)	**2.218E-02(3.2E-02)**
Model30-2	1.119E-01(6.4E-02)†	1.194E-01(8.7E-02)†	8.149E-02(1.0E-01)†	1.531E-01(1.1E-01)†	**2.495E-01(7.7E-02)**†	1.875E-01(7.4E-02)
Model30-3	5.385E-03(2.5E-03)†	5.367E-03(1.7E-03)†	4.209E-03(7.8E-03)†	5.832E-03(3.1E-03)†	2.695E-02(2.6E-02)†	**3.943E-02(1.8E-02)**
Model30-4	1.157E-02(4.8E-03)†	1.241E-02(5.8E-03)†	6.370E-03(4.7E-03)†	1.346E-02(6.1E-03)†	3.140E-02(1.3E-02)	**3.466E-02(1.0E-02)**
Model30-5	1.419E-04(2.8E-04)†	1.246E-04(3.5E-04)†	8.739E-05(2.7E-04)†	9.410E-05(1.9E-04)†	2.466E-03(3.7E-03)	**2.783E-03(2.7E-03)**
Model30-6	9.242E-02(3.0E-02)†	1.091E-01(3.1E-02)†	5.308E-02(4.3E-02)†	1.042E-01(2.7E-02)†	1.486E-01(4.6E-02)†	**2.057E-01(4.4E-02)**
Model30-7	4.776E-02(1.4E-01)†	7.170E-02(1.3E-01)†	3.716E-02(1.3E-01)†	3.126E-02(3.4E-02)†	5.487E-01(2.2E-01)	**5.731E-01(2.9E-01)**
Model30-8	9.289E-03(4.2E-02)†	2.046E-03(8.2E-03)†	1.228E-04(5.6E-04)†	1.075E-03(2.7E-03)†	**7.343E-02(9.6E-02)**	4.512E-02(5.8E-02)
Model30-9	1.396E-01(1.0E-01)†	1.016E-01(4.0E-02)†	7.748E-02(6.8E-02)†	1.007E-01(3.0E-02)†	2.788E-01(1.4E-01)	**3.230E-01(1.2E-01)**
Model30-10	8.608E-02(4.4E-02)†	7.205E-02(2.6E-02)†	4.341E-02(2.6E-02)†	7.215E-02(2.6E-02)†	**2.214E-01(1.7E-01)**	1.970E-01(8.5E-02)

Table 14. HV of the Six Algorithms on the 10 Random Models with 50 Features

Problem	NSGA-II	IBEA	MOEAD	SPEA2+SDE	PAES	GrES
Model50-1	1.230E-01(9.1E-03)†	1.368E-01(3.9E-02)†	1.005E-01(5.1E-02)†	1.289E-01(2.6E-02)†	2.073E-01(6.9E-02)†	**2.481E-01(6.7E-02)**
Model50-2	1.301E-01(1.1E-01)†	7.882E-02(9.0E-02)†	3.174E-02(6.0E-02)†	1.174E-01(8.1E-02)†	4.278E-01(1.8E-01)	**5.370E-01(2.2E-01)**
Model50-3	2.518E-03(2.8E-03)†	9.719E-03(2.0E-02)†	1.474E-03(2.6E-03)†	9.195E-03(2.2E-02)†	2.179E-01(2.2E-01)	**2.707E-01(2.1E-01)**
Model50-4	3.137E-01(3.1E-01)†	2.560E-01(2.7E-01)†	4.128E-02(1.1E-01)†	2.266E-01(3.0E-01)†	8.540E-01(9.1E-02)†	**9.165E-01(7.6E-02)**
Model50-5	1.232E-01(1.1E-01)†	1.213E-01(9.4E-02)†	3.926E-02(3.4E-02)†	9.377E-02(4.5E-02)†	4.337E-01(1.3E-01)†	**5.901E-01(1.9E-01)**
Model50-6	4.675E-03(3.0E-03)†	4.645E-03(3.4E-03)†	1.066E-03(1.9E-03)†	5.180E-03(2.9E-03)†	9.887E-03(7.2E-03)†	**1.517E-02(3.0E-03)**
Model50-7	6.250E-02(1.1E-01)†	4.729E-02(1.0E-01)†	2.674E-02(6.2E-02)†	8.568E-02(1.2E-01)†	4.162E-01(1.2E-01)	**4.768E-01(6.9E-02)**
Model50-8	2.991E-01(6.1E-02)†	3.032E-01(4.5E-02)†	1.188E-01(9.6E-02)†	2.979E-01(5.4E-02)†	4.933E-01(5.4E-02)	**5.056E-01(5.1E-02)**
Model50-9	5.407E-02(4.6E-02)†	6.949E-02(5.5E-02)†	1.194E-02(1.9E-02)†	4.831E-02(2.3E-02)†	3.751E-01(2.0E-01)	**4.272E-01(1.6E-01)**
Model50-10	1.428E-01(6.8E-02)†	1.476E-01(7.1E-02)†	8.360E-02(9.0E-02)†	1.243E-01(4.5E-02)†	**5.059E-01(2.1E-01)**	4.933E-01(1.3E-01)

Table 15. HV of the Six Algorithms on the 10 Random Models with 100 Features

Problem	NSGA-II	IBEA	MOEAD	SPEA2+SDE	PAES	GrES
Model100-1	7.177E-03(2.5E-02)†	7.520E-03(3.0E-02)†	5.778E-04(3.1E-03)†	8.135E-03(2.9E-02)†	2.863E-01(1.9E-01)†	**5.076E-01(1.9E-01)**
Model100-2	1.751E-01(2.0E-01)†	1.520E-01(2.2E-01)†	1.144E-02(6.3E-02)†	1.996E-01(2.0E-01)†	4.371E-01(1.7E-01)†	**6.737E-01(1.2E-01)**
Model100-3	1.191E-01(1.8E-01)†	3.893E-02(1.2E-01)†	2.133E-02(6.8E-02)†	1.476E-01(2.1E-01)†	7.914E-01(2.4E-01)†	**9.572E-01(6.8E-02)**
Model100-4	5.556E-02(1.5E-01)†	5.323E-02(1.5E-01)†	2.074E-02(7.9E-02)†	5.252E-02(1.5E-01)†	6.694E-01(1.4E-01)†	**8.153E-01(8.4E-02)**
Model100-5	2.778E-02(4.7E-02)†	2.267E-02(4.7E-02)†	2.156E-03(1.1E-02)†	1.768E-02(4.0E-02)†	1.750E-01(7.2E-02)†	**2.087E-01(8.1E-02)**
Model100-6	4.589E-03(1.1E-02)†	9.446E-03(2.0E-02)†	3.542E-04(1.1E-03)†	5.354E-03(1.3E-02)†	**2.923E-01(2.8E-01)**	2.519E-01(1.8E-01)
Model100-7	3.152E-02(7.8E-02)†	6.004E-02(1.1E-01)†	6.416E-03(2.1E-02)†	6.840E-02(1.0E-01)†	5.659E-01(6.3E-02)†	**6.107E-01(3.7E-02)**
Model100-8	3.391E-02(7.1E-02)†	6.581E-02(9.9E-02)†	5.686E-02(1.2E-01)†	4.771E-02(7.6E-02)†	8.018E-01(3.0E-01)†	**1.199E+00(1.7E-01)**
Model100-9	1.666E-01(1.9E-01)†	1.804E-01(1.9E-01)†	3.355E-02(9.2E-02)†	9.929E-02(1.7E-01)†	6.690E-01(1.7E-01)†	**8.971E-01(8.2E-02)**
Model100-10	3.663E-04(1.1E-03)†	3.752E-03(1.6E-02)†	4.087E-05(2.2E-04)†	4.930E-05(2.4E-04)†	1.640E-01(6.8E-02)	**1.802E-01(4.9E-02)**

case in 7 out of 10 experiments with 30 features and 9 out of 10 cases in each of the 2 other experiments (with 50 and 100 features). Interestingly, although GrES had higher HV values in the same number of experiments with models with 50 and 100 features, only four of these differences were statistically significant for models with 50 features, while all of the differences were statistically significant for models with 100 features.

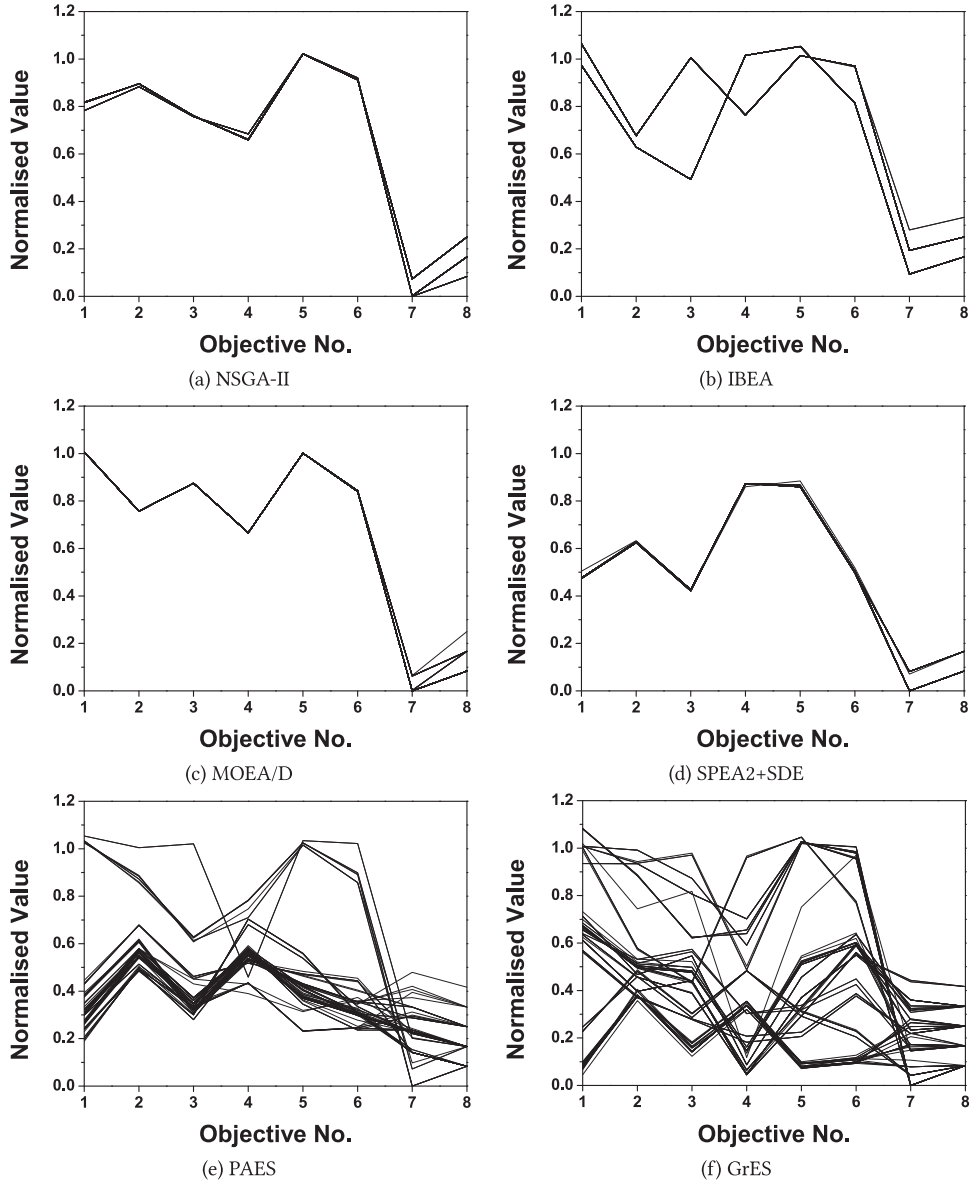

Fig. 3. The final population of one run of the six algorithms on random model Model100-1, shown by the parallel coordinates plot [46]. In each figure, the first six objectives are prioritisation objectives and the last two are cost and size objectives; the first objective pairwise coverage is not shown as all the solutions achieve the same value.

For a visual understanding of the solutions' distribution and also of what a higher HV means, in Figure 3 we use parallel coordinates [46] to plot the final population of one typical run on the random model Model100-1. Parallel coordinates map a set of solutions in a high-dimensional space onto a 2D graph. A point is represented as a polyline connecting the values of the objectives. Parallel coordinates can reflect the convergence, coverage and uniformity to some extent [46]. As we

Table 16. HV and [min,max] Raw Objective Values of the Populations Obtained by the Six Algorithms in Figure 3

Algorithm	HV	Objectives								
		PC	NF	FS	NC	VCCC	CoC	D	TSC	TSS
NSGA-II	1.8E-04	18429	[558.5, 561.8]	[352.2, 353.1]	[340.0, 340.2]	[27.2, 27.3]	[166.6, 166.8]	[68.3, 68.4]	[**8229.1**, 8429.8]	[**49**, 51]
IBEA	5.1E-05	18429	[509.0, 544.0]	[366.9, 370.0]	[326.0, 353.8]	[25.3, 26.8]	[161.9, 168.0]	[67.9, 69.3]	[8485.0, 8993.3]	[50, 52]
MOEA/D	9.4E-05	18429	[537.6, 538.3]	[361.4, 361.5]	[334.2, 334.2]	[27.3, 27.3]	[170.1, 170.1]	[69.0, 69.0]	[**8229.1**, 8406.4]	[**49**, 51]
SPEA2+SDE	7.8E-03	18429	[587.8, 590.6]	[369.7, 370.3]	[357.2, 357.5]	[26.3, 26.3]	[171.8, 172.1]	[71.0, 72.2]	[**8229.1**, 8452.5]	[**49**, 50]
PAES	2.6E-01	18429	[523.8, 617.1]	[343.8, 379.7]	[321.2, 364.7]	[26.7, 28.5]	[164.9, 179.5]	[66.1, 74.6]	[**8229.1**, 9539.9]	[**49**, 53]
GrES	6.6E-01	18429	[500.2, **628.6**]	[345.8, **388.2**]	[329.3, **372.8**]	[25.8, **30.2**]	[163.0, **181.3**]	[67.2, **75.9**]	[**8229.1**, 9441.0]	[**49**, 53]

Overall maximum and minimum values are highlighted in boldface in those objectives being maximised and minimised, respectively. PC: Pairwise coverage, NF: Number of faults, FS: Feature size, NC: Number of changes, VCCC: Variability coverage and cyclomatic complexity, CoC: Coefficient of connectivity-density, D: Dissimilarity, TSC: Test suite cost, TSS: Test suite size.

Table 17. HV of the Six Algorithms on the 19 Realistic Models

Problem	NSGA-II	IBEA	MOEAD	SPEA2+SDE	PAES	GrES
Apache	0.000E+00(0.0E+00)†	0.000E+00(0.0E+00)†	0.000E+00(0.0E+00)†	0.000E+00(0.0E+00)†	**5.365E-02(8.7E-02)**	3.417E-02(7.3E-02)
argo-uml-spl	2.294E-05(7.7E-06)	2.294E-05(7.7E-06)	**2.594E-05(2.1E-05)**	2.294E-05(7.7E-06)	2.322E-05(8.5E-06)	2.322E-05(7.6E-06)
BerkeleyDB	6.774E-05(6.0E-05)	5.266E-05(5.1E-05)	7.181E-05(6.4E-05)	5.438E-05(4.5E-05)	**8.389E-05(7.5E-05)**	7.753E-05(7.2E-05)
BerkeleyDBFootprint	3.996E-03(1.4E-03)†	4.604E-03(3.6E-03)†	3.632E-03(1.6E-04)†	3.637E-03(1.5E-04)†	**1.628E-02(2.2E-02)**†	9.718E-03(1.1E-02)
BerkeleyDBMemory	1.973E-03(7.5E-03)†	1.295E-03(1.8E-03)†	2.752E-04(8.5E-04)†	6.458E-04(1.8E-03)†	**2.313E-01(1.8E-01)**†	1.232E-01(1.0E-01)
BerkeleyDBPerformance	4.133E-02(1.8E-02)†	4.786E-02(3.2E-02)†	2.581E-02(2.3E-02)†	4.411E-02(3.2E-02)†	1.410E-01(6.9E-02)	**1.522E-01(5.2E-02)**
Curl	4.587E-02(4.6E-02)†	4.495E-02(3.5E-02)†	2.568E-02(3.0E-02)†	4.115E-02(3.4E-02)†	1.106E-01(7.2E-02)	**1.263E-01(4.4E-02)**
DesktopSearcher	3.809E-02(1.8E-02)†	3.631E-02(2.0E-02)†	2.729E-02(2.0E-02)†	4.005E-02(3.3E-02)†	1.178E-01(4.7E-02)	**1.196E-01(6.0E-02)**
fame-dbms-fm	2.655E-01(7.1E-02)†	2.624E-01(6.7E-02)†	1.476E-01(1.1E-01)†	2.693E-01(8.9E-02)†	3.482E-01(6.3E-02)†	**4.048E-01(2.9E-02)**
gpl	1.273E-04(9.0E-05)†	1.220E-04(8.8E-05)†	6.888E-05(8.3E-05)†	1.159E-04(8.6E-05)†	4.016E-04(1.1E-04)	**4.269E-04(1.0E-04)**
LinkedList	1.925E-02(8.7E-03)†	1.892E-02(8.5E-03)†	1.179E-02(1.1E-02)†	1.567E-02(6.4E-03)†	**3.122E-02(2.2E-02)**	2.950E-02(1.4E-02)
LLVM	4.301E-02(1.4E-02)†	4.274E-02(2.0E-02)†	2.868E-02(1.8E-02)†	4.733E-02(1.4E-02)†	1.097E-01(1.1E-01)	**1.234E-01(1.6E-01)**
PKJab	1.351E-03(1.9E-03)†	2.816E-03(7.2E-03)†	3.722E-03(7.2E-03)†	3.929E-03(7.6E-03)†	**5.911E-02(1.1E-01)**	5.550E-02(1.1E-01)
Prevayler	4.671E-02(3.4E-02)	5.144E-02(4.3E-02)	2.150E-02(2.6E-02)†	**5.515E-02(5.0E-02)**	4.923E-02(5.3E-02)	5.039E-02(5.9E-02)
SensorNetwork	5.434E-02(1.9E-02)†	5.365E-02(2.0E-02)†	3.045E-02(2.6E-02)†	5.427E-02(1.7E-02)†	9.438E-02(4.0E-02)	**1.009E-01(7.3E-02)**
TankWar	6.673E-02(4.3E-02)†	7.917E-02(5.5E-02)†	1.838E-02(3.3E-02)†	5.407E-02(5.0E-02)†	1.822E-01(9.0E-02)†	**3.395E-01(1.6E-01)**
Wget	5.463E-03(1.0E-02)†	2.581E-03(7.4E-04)†	5.242E-03(1.1E-02)†	5.770E-03(1.1E-02)†	**4.003E-02(1.1E-01)**	1.968E-02(2.1E-02)
x264	4.084E-02(4.1E-03)†	3.992E-02(4.5E-03)†	2.541E-02(1.5E-02)†	4.182E-02(3.3E-03)†	4.558E-02(9.5E-03)†	**5.057E-02(5.4E-03)**
ZipMe	1.838E-02(3.1E-02)†	1.782E-02(3.0E-02)†	1.521E-02(3.2E-02)†	2.081E-02(3.3E-02)†	3.930E-02(5.5E-02)	**4.124E-02(5.3E-02)**

can see in Figure 3, NSGA-II, IBEA, MOEA/D, and SPEA2+SDE fail to maintain diversity, with their solutions concentrated in a tiny area. In contrast, PAES and GrES are able to find solutions within different regions, providing multiple options for the decision-maker. If we compare GrES with PAES, it appears that GrES performs better than PAES in terms of both convergence and diversity, particularly on the prioritisation objectives (objectives no. 1–6), with more diverse solutions being obtained. In addition, for practical understanding, in Table 16, we give the range of raw objective values of the six populations shown in Figure 3 and their HV results. As can be seen in the table, GrES achieves the maximum values in six prioritisation objectives and shares the minimum values in the two cost objectives.

Table 17 gives the results with realistic models. Again, the best technique tended to be one of PAES and GrES; this was the case in 17 of the 19 experiments. GrES was also best for the case (Drupal) where we had real attribute values (Table 18). Almost all the differences between

Table 18. HV of the Six Algorithms on the Real Model DrupalFM

Problem	NSGA-II	IBEA	MOEAD	SPEA2+SDE	PAES	GrES
DrupalFM	3.195E-01(9.6E-02)†	3.020E-01(4.7E-02)†	2.220E-01(1.3E-01)†	2.695E-01(9.8E-02)†	6.731E-01(2.2E-01)	**6.789E-01(2.2E-01)**

Table 19. HV of NSGA-II$_H$ and GrES on the 2 Coverage and Cost Objectives on the 10 Random Models with 30 Features

Problem	NSGA-II$_H$	GrES
Model30-1	1.009E+00(6.8E-16)	**1.018E+00(2.8E-02)**
Model30-2	**1.021E+00(8.4E-02)**†	7.583E-01(8.6E-02)
Model30-3	**1.073E+00(4.2E-02)**†	1.027E+00(3.7E-02)
Model30-4	**1.008E+00(6.5E-02)**	9.958E-01(5.8E-02)
Model30-5	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model30-6	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model30-7	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model30-8	**1.033E+00(4.8E-02)**†	1.003E+00(1.8E-02)
Model30-9	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model30-10	**1.031E+00(3.8E-02)**	1.028E+00(3.6E-02)

GrES/PAES and the genetic algorithms were statistically significant. However, very few of the differences between GrES and PAES were statistically significant.

Overall, it seems that evolution strategies lead to higher HV values than the genetic algorithms. In fact, all four types of genetic algorithms (whether based on Pareto dominance or dedicated for many-objective optimisation) produced similar HV values, as their difference lies in how to select solutions for survival (i.e., forming the next population), which does not matter much for the considered problem. What matters here is how to select solutions for variation (i.e., producing offspring). In this regard, continually doing variations around the seed individual and its offspring differentiates PAES and GrES from the others. In addition, GrES tended to lead to higher HV values than PAES. The randomly generated models provided us with the opportunity to see how patterns develop as the model size increases (all other parameters were fixed). It seems that the improvement that GrES provided over PAES was greater for the larger models. Most likely, this is because the selection operation, with the help of problem-specific knowledge, has a greater chance of producing promising offspring, particularly for problems with a larger search space.

5.3 Research Question 3: Comparison with Existing Approaches on Specific Objectives

The two previously proposed approaches, NSGA-II$_H$ and NSGA-II$_P$, used different sets of objectives and there is the question of how GrES compares with these approaches if we restrict attention to the corresponding sets of objectives. Thus, we performed two additional sets of experiments. The first compared GrES with N SGA-II$_H$in experiments that only used the objectives originally used in the work by Lopez-Herrejon et al. [48] (coverage and cost). The other compared GrES with NSGA-II$_P$ using the corresponding set of six prioritisation objectives [57].

Tables 19, 20, and 21 give the results regarding NSGA-II$_H$ for randomly generated models and Table 22 contains the results for the realistic models; in all cases, the comparison was on only two objectives. In the experiment, NSGA-II$_H$ generally led to higher HV values than GrES on the models with 30 features, but was worse on large models (e.g., with 100 features). For the rest,

Table 20. HV of NSGA-II$_H$ and GrES on the 2 Coverage and Cost Objectives on the 10 Random Models with 50 Features

Problem	NSGA-II$_H$	GrES
Model50-1	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-2	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-3	1.047E+00(4.5E-16)†	**1.060E+00(2.3E-02)**
Model50-4	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-5	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-6	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-7	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-8	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-9	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model50-10	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)

Table 21. HV of NSGA-II$_H$ and GrES on the 2 Coverage and Cost Objectives on the 10 Random Models with 100 Features

Problem	NSGA-II$_H$	GrES
Model100-1	1.064E+00(9.4E-03)†	**1.080E+00(3.8E-03)**
Model100-2	1.099E+00(4.8E-03)	**1.100E+00(4.5E-16)**
Model100-3	1.030E+00(2.7E-02)†	**1.072E+00(5.2E-03)**
Model100-4	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model100-5	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model100-6	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model100-7	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
Model100-8	1.062E+00(1.5E-02)†	**1.100E+00(4.5E-16)**
Model100-9	1.099E+00(5.7E-03)	**1.100E+00(4.5E-16)**
Model100-10	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)

they had similar HV values. There were almost no statistically significant differences. The two techniques had identical HV values on the Drupal model (Table 23).

Tables 24 and 25 give the results regarding NSGA-II$_P$ for randomly generated models and Tables 26 and 27 contain the results for the realistic models (all compared on six objectives). We did not perform experiments with models that had 100 features since the computation time required by NSGA-II$_P$ was excessive. In these experiments, NSGA-II$_P$ led to slightly higher HV values than GrES on the models with 30 features, but was generally worse on the other models. In contrast to the experiments with two objectives, almost all of the results were statistically significant.

Overall, the proposed GrES technique had slightly lower HV values than the other techniques when used with smaller models but tended to have higher HV values with larger models. Very few of the differences were statistically significant when we considered two objectives but most were statistically significant when we compared on six objectives.

One probable explanation is that, for small models (30 features), these existing techniques are already competent and their focus on fewer specific objectives makes them obtain better results than GrES, which attempts to optimise over the nine objectives but is then evaluated on a subset of these. For large models, incorporating the problem knowledge into the design of the search algorithm can largely improve its performance (HV), particularly considering the pairwise coverage first which can narrow down the search space significantly. Recall also that GrES was

Table 22. HV of NSGA-II$_H$ and GrES on the 2 Coverage and Cost Objectives on the 19 Realistic Models

Problem	NSGA-II$_H$	GrES
Apache	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
argo-uml-spl	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
BerkeleyDB	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
BerkeleyDBFootprint	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
BerkeleyDBMemory	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
BerkeleyDBPerformance	**9.792E-01(2.3E-02)**	9.750E-01(4.5E-16)
Curl	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
DesktopSearcher	9.667E-01(7.4E-02)	9.667E-01(6.4E-02)
fame-dbms-fm	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
gpl	1.090E+00(3.1E-02)	**1.093E+00(2.5E-02)**
LinkedList	1.021E+00(3.1E-02)	**1.024E+00(3.4E-02)**
LLVM	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
PKJab	**9.667E-01(6.8E-02)**	9.500E-01(5.1E-02)
Prevayler	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)
SensorNetwork	**1.010E+00(4.0E-02)**	1.003E+00(1.8E-02)
TankWar	**1.036E+00(2.9E-02)**	1.028E+00(2.0E-02)
Wget	1.033E+00(4.1E-02)	**1.042E+00(4.5E-02)**
x264	1.098E+00(1.1E-02)	**1.100E+00(4.5E-16)**
ZipMe	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)

Table 23. HV of NSGA-II$_H$ and GrES on the 2 Coverage and Cost Objectives on the Real Model DrupalFM

Problem	NSGA-II$_H$	GrES
DrupalFM	1.100E+00(4.5E-16)	1.100E+00(4.5E-16)

Table 24. HV of NSGA-II$_P$ and GrES on the 6 Prioritisation Objectives on the 10 Random Models with 30 Features

Problem	NSGA-II$_P$	GrES
Model30-1	**3.552E-01(7.2E-01)**†	0.000E+00(0.0E+00)
Model30-2	**5.352E-02(2.0E-01)**†	0.000E+00(0.0E+00)
Model30-3	**5.905E-02(3.2E-01)**	0.000E+00(0.0E+00)
Model30-4	**2.293E-02(5.8E-02)**†	0.000E+00(0.0E+00)
Model30-5	**5.905E-02(3.2E-01)**	0.000E+00(0.0E+00)
Model30-6	**5.395E-03(2.9E-02)**†	7.217E-05(1.2E-04)
Model30-7	0.000E+00(0.0E+00)†	**3.823E-01(3.1E-01)**
Model30-8	2.928E-03(1.6E-02)†	**9.289E-03(2.3E-02)**
Model30-9	3.723E-05(2.0E-04)†	**1.916E-02(4.3E-02)**
Model30-10	9.667E-07(4.3E-06)†	**1.913E-01(2.7E-01)**

Table 25. HV of NSGA-II$_P$ and GrES on the 6 Prioritisation Objectives on the 10 Random Models with 50 Features

Problem	NSGA-II$_P$	GrES
Model50-1	**5.369E-03(2.9E-02)**†	4.372E-04(4.4E-04)
Model50-2	0.000E+00(0.0E+00)†	**2.478E-01(2.4E-01)**
Model50-3	0.000E+00(0.0E+00)†	**8.311E-02(1.2E-01)**
Model50-4	0.000E+00(0.0E+00)†	**3.510E-01(1.2E-01)**
Model50-5	0.000E+00(0.0E+00)†	**1.909E-01(1.9E-01)**
Model50-6	**5.905E-02(3.2E-01)**	0.000E+00(0.0E+00)
Model50-7	0.000E+00(0.0E+00)†	**3.495E-01(1.2E-01)**
Model50-8	1.319E-02(8.1E-03)†	**4.235E-01(5.9E-02)**
Model50-9	1.537E-05(7.7E-05)†	**1.269E-01(1.2E-01)**
Model50-10	6.560E-05(1.3E-04)†	**2.071E-01(1.4E-01)**

Table 26. HV of NSGA-II$_P$ and GrES on the 6 Prioritisation Objectives on the 19 Realistic Models

Problem	NSGA-II$_P$	GrES
Apache	0.000E+00(0.0E+00)†	**1.476E+00(6.7E-01)**
argo-uml-spl	6.496E-01(8.7E-01)†	**1.653E+00(4.5E-01)**
BerkeleyDB	**1.713E+00(3.2E-01)**†	0.000E+00(0.0E+00)
BerkeleyDBFootprint	0.000E+00(0.0E+00)†	**1.240E+00(8.3E-01)**
BerkeleyDBMemory	1.817E-04(9.9E-04)†	**2.303E-01(1.9E-01)**
BerkeleyDBPerformance	**3.062E-02(6.3E-02)**	8.433E-06(8.8E-06)
Curl	**1.057E-01(1.0E-01)**†	6.000E-07(1.2E-06)
DesktopSearcher	**8.858E-01(9.0E-01)**†	0.000E+00(0.0E+00)
fame-dbms-fm	2.834E-02(1.5E-01)†	**1.144E-01(1.9E-02)**
gpl	**1.975E-02(3.4E-02)**†	0.000E+00(0.0E+00)
LinkedList	**2.362E-01(6.1E-01)**†	0.000E+00(0.0E+00)
LLVM	0.000E+00(0.0E+00)†	**5.315E-01(8.3E-01)**
PKJab	4.724E-01(8.0E-01)†	**1.713E+00(3.2E-01)**
Prevayler	5.315E-01(8.3E-01)†	**1.772E+00(1.1E-15)**
SensorNetwork	2.424E-03(2.9E-03)†	**2.687E-01(4.5E-01)**
TankWar	4.000E-07(1.5E-06)†	**1.494E-01(2.3E-01)**
Wget	**2.325E-02(5.6E-02)**†	2.813E-05(8.0E-05)
x264	2.806E-02(5.0E-02)†	**4.233E-02(1.0E-02)**
ZipMe	3.000E-07(4.7E-07)†	**1.772E+00(1.1E-15)**

Table 27. HV of NSGA-II$_P$ and GrES on the 6 Prioritisation Objectives on the Real Model DrupalFM

Problem	NSGA-II$_P$	GrES
DrupalFM	0.000E+00(0.0E+00)†	**1.315E-01(1.1E-01)**

optimising on all of the objectives but then the comparison was only on the objectives used by the other approaches; this clearly disadvantages GrES.

5.4 Research Question 4: How Large Are the Differences in HV Values in the Experiments Used for RQ1-3?

To further investigate the differences between the HV values of the populations returned by different algorithms, the Vargha and Delaney's $\widehat{A_{12}}$ statistic [69] was used to evaluate the effect size, i.e., determine which technique led to higher HV values and to what extent. The $\widehat{A_{12}}$ statistic generalises the notion of effect size estimator to data that does not follow a normal distribution, such as those generated by the techniques evaluated in the experiments. Specifically, the $\widehat{A_{12}}$ statistic shown in each cell of Table 28 relates to how often on average GrES provides higher HV values than the technique in the column for the model in the row. The value of the $\widehat{A_{12}}$ statistic ranges from 0.0 to 1.0, and when the value is exactly 0.5, it means that either the technique provided exactly the same results as GrES on all runs, or that GrES provided higher HV values for exactly 50% of the runs. If the value of the $\widehat{A_{12}}$ statistic is lower than 0.5, the technique provided higher HV values than GrES for a majority of runs. Conversely, if the value is higher than 0.5, GrES provided higher HV values for a majority of runs. Vargha and Delaney [69] suggested the use of thresholds for interpreting the effect size: values around 0.5 (0.43 – 0.57) means a negligible difference; values over 0.57 indicates a small (0.57 – 0.64), medium (0.65 – 0.71), or large (0.72 – 1) difference in favour of GrES; values below 0.43 indicate a small (0.43 – 0.36), medium (0.36 – 0.29), or large (0.29 – 0.0) difference in favour of the algorithm specified in the column of the table. Cells indicating medium, large, and very large differences are shaded in light grey, grey, and dark grey, respectively.

The 9-objectives HV values confirm the superiority of GrES for 9 objectives, showing large differences in its favour in 285 out of 340 pairwise comparisons. Furthermore, there is not a single cell showing large nor even medium differences against GrES for the peers except PAES. Comparing GrES with PAES, large, medium, and small differences in favour of GrES vs PAES are 13 vs 1, 5 vs 2, and 13 vs 2, respectively.

The $\widehat{A_{12}}$ results for the HV computed using 6 and 2 objectives are also shown in the last columns of Table 28. Specifically, the second to last column shows the $\widehat{A_{12}}$ comparison with the NSGA-II$_P$ algorithm for the HV of the six objectives used in [57]. The results show the superiority of GrES also in this case, with large differences in its favour for 31 out of 40 models, and with negligible differences in 4 out of the 9 remaining comparisons. The last column shows the $\widehat{A_{12}}$ comparison with the NSGA-II$_H$ algorithm for the HV of the two objectives used in [48] (the NSGA-II$_H$ algorithm was re-executed using only those two objectives). The results show that the HV values are quite similar for the 2-objectives case, showing negligible differences in 43 out of the 50 models, with large differences in favour of GrES in 3 large models (100 features), and large differences in favour of NSGA-II$_H$ in 2 small models (30 features).

5.5 Research Question 5: How Long Do the Different Algorithms Take?

Execution time information is given in Table 29 and Figure 4. Specifically, Table 29 provides the average execution time in seconds per model (in rows) and algorithm (in columns). Figure 4 shows a scatterplot of the average execution time (Y axis measured in seconds) per model size (X axis measured in features). Thus, the values in the horizontal axis of Figure 4 range from a minimum of 6 features for *Prevlayer* to a maximum of 117 features for *BerkeleyDatabase*. Each point in the figure represents the run of an algorithm (represented using a specific shade of grey and line dash) for a number

Table 28. $\widehat{A_{12}}$ Values Comparing the Hyper-Volumes of the Fronts Generated by GrES with Those Generated by the Other Algorithms

Problem	9 Objectives HV							6 Obj. HV	2 Obj. HV
	NSGAII$_H$	IBEA	MOEAD	NSGAII	PAES	NSGAII$_P$	SPEA2+SDE	NSGAII$_P$	NSGAII$_H$
30Feat-10CTC1	**1.00**	.895	.979	.872	.632	.727	.902	.300	.550
30Feat-10CTC2	**.997**	.776	.876	.842	.273	.936	.710	.317	.026
30Feat-10CTC3	**1.00**	**1.00**	.978	**1.00**	.751	.967	**1.00**	.483	.250
30Feat-10CTC4	**1.00**	.991	**1.00**	.998	.631	.752	.981	.350	.452
30Feat-10CTC5	**1.00**	.973	.986	.978	.619	.933	.991	.467	.500
30Feat-10CTC6	**1.00**	.997	**1.00**	.990	.828	.967	**1.00**	.933	.500
30Feat-10CTC7	**1.00**	.961	.977	.973	.501	**1.00**	.998	**1.00**	.500
30Feat-10CTC8	**1.00**	.968	.998	.943	.434	.983	.978	.967	.350
30Feat-10CTC9	**1.00**	.983	.979	.909	.612	**1.00**	.993	.948	.500
30Feat-10CTC10	**1.00**	.958	.984	.912	.527	**1.00**	.958	**1.00**	.484
50Feat-10CTC1	**1.00**	.953	.984	.993	.670	**1.00**	.980	.967	.500
50Feat-10CTC2	**1.00**	.999	**1.00**	.973	.647	**1.00**	**1.00**	**1.00**	.500
50Feat-10CTC3	**1.00**	.992	**1.00**	**1.00**	.607	**1.00**	.990	**1.00**	.617
50Feat-10CTC4	**1.00**	**1.00**	**1.00**	**1.00**	.693	**1.00**	**1.00**	**1.00**	.500
50Feat-10CTC5	**1.00**	.987	**1.00**	.989	.733	**1.00**	**1.00**	**1.00**	.500
50Feat-10CTC6	**1.00**	.999	**1.00**	.990	.819	.967	.998	.483	.500
50Feat-10CTC7	**1.00**	**1.00**	**1.00**	**1.00**	.637	**1.00**	**1.00**	**1.00**	.500
50Feat-10CTC8	**1.00**	**1.00**	**1.00**	.996	.543	**1.00**	.999	**1.00**	.500
50Feat-10CTC9	**1.00**	.989	**1.00**	.992	.591	**1.00**	**1.00**	**1.00**	.500
50Feat-10CTC10	**1.00**	.991	.986	.992	.499	**1.00**	.998	**1.00**	.500
100Feat-10CTC1	**1.00**	.999	**1.00**	.999	.808	-	.999	-	.871
100Feat-10CTC2	**1.00**	.972	.999	.982	.911	-	.979	-	.517
100Feat-10CTC3	**1.00**	**1.00**	**1.00**	**1.00**	.756	-	**1.00**	-	.952
100Feat-10CTC4	**1.00**	.999	**1.00**	.997	.808	-	**1.00**	-	.500
100Feat-10CTC5	**1.00**	.994	**1.00**	.993	.684	-	.997	-	.500
100Feat-10CTC6	**1.00**	**1.00**	**1.00**	**1.00**	.528	-	**1.00**	-	.500
100Feat-10CTC7	**1.00**	**1.00**	**1.00**	**1.00**	.740	-	**1.00**	-	.500
100Feat-10CTC8	**1.00**	**1.00**	**1.00**	**1.00**	.897	-	**1.00**	-	.983
100Feat-10CTC9	**1.00**	**1.00**	**1.00**	**1.00**	.899	-	**1.00**	-	.517
100Fea-10CTC10	**1.00**	.998	**1.00**	**1.00**	.597	-	**1.00**	-	.500
Apache	.733	.733	.733	.733	.446	.733	.733	**1.00**	.500
argo-uml-spl	.000	.514	.519	.514	.513	.662	.514	**.803**	.500
BerkeleyDB	**.848**	.545	.476	.525	.466	.681	.474	.000	.500
BerkeleyDBFp	**1.00**	.776	.832	.761	.341	.992	.821	.950	.500
BerkeleyDBMem	**1.00**	**1.00**	**1.00**	.996	.312	**1.00**	**1.00**	**1.00**	.500
BerkeleyDBPerf	**1.00**	.958	.990	.994	.579	.964	.972	.530	.483
Curl	**1.00**	.951	.970	.942	.610	.933	.943	.230	.500
DesktopSearcher	**1.00**	.974	.986	.979	.490	**1.00**	.950	.217	.499
fame_dbms_fm	**1.00**	**1.00**	**1.00**	**1.00**	.830	**1.00**	**1.00**	.967	.500
gpl	**1.00**	.982	.993	.978	.543	.440	.982	.183	.517
LinkedList	.999	.928	.941	.871	.562	**1.00**	.971	.417	.517
LLVM	**1.00**	.966	.980	.992	.572	**1.00**	.990	.950	.500
PKJab	**.996**	.887	.873	.908	.424	.951	.882	.850	.450
Prevayler	**.986**	.453	.774	.442	.485	.972	.454	.850	.500
SensorNetwork	**1.00**	.946	.993	.941	.508	**1.00**	.953	.922	.466
TankWar	**1.00**	.983	**1.00**	**1.00**	.814	**1.00**	.993	**1.00**	.450
Wget	**1.00**	.992	.923	.946	.414	.937	.924	.644	.550
x264	**1.00**	**1.00**	**1.00**	.998	.674	.816	**1.00**	.800	.517
ZipMe	**1.00**	.712	.758	.696	.569	**1.00**	.667	**1.00**	.500
DrupalFM	**1.00**	.992	.988	.973	.510	**1.00**	.998	**1.00**	.500

Table 29. Average Execution Times of the Algorithms per Model Using Nine Objectives (in Seconds)

Problem	Algorithms							
	GrES	NSGA-II$_H$	IBEA	MOEAD	NSGA-II	PAES	NSGA-II$_P$	SPEA2+SDE
30Feat-10CTC1	7.76	10.55	24.52	8.27	8.05	9.50	124.27	11.10
30Feat-10CTC2	6.08	6.78	23.22	6.60	6.85	6.82	88.41	9.41
30Feat-10CTC3	7.54	9.26	24.23	7.49	7.58	8.92	105.90	10.77
30Feat-10CTC4	4.57	5.06	26.51	4.89	5.18	5.19	58.34	14.61
30Feat-10CTC5	8.31	10.38	23.80	7.99	8.01	9.63	206.14	14.07
30Feat-10CTC6	8.00	9.50	24.40	7.68	7.63	9.54	203.61	10.39
30Feat-10CTC7	9.89	11.95	26.64	9.81	9.69	11.54	167.77	12.10
30Feat-10CTC8	7.27	9.02	25.87	7.54	7.38	8.86	432.87	9.61
30Feat-10CTC9	8.28	10.22	25.46	8.24	8.15	9.89	161.55	11.06
30Feat-10CTC10	9.80	12.16	27.26	9.74	9.56	11.19	188.15	12.00
50Feat-10CTC1	14.83	19.83	30.08	13.38	13.50	17.87	590.89	16.39
50Feat-10CTC2	32.48	40.49	47.37	30.74	30.78	38.13	1918.98	32.84
50Feat-10CTC3	40.11	52.44	55.03	39.14	39.26	45.24	2171.35	41.43
50Feat-10CTC4	33.00	42.71	47.11	31.93	31.89	36.83	1850.81	33.91
50Feat-10CTC5	25.15	32.72	42.54	24.72	24.58	27.81	1315.45	26.40
50Feat-10CTC6	35.41	44.67	47.91	34.08	34.21	39.16	1614.20	36.66
50Feat-10CTC7	36.25	47.24	51.62	35.73	35.67	38.50	1781.94	38.15
50Feat-10CTC8	27.47	35.22	41.63	26.01	26.07	30.03	1269.60	28.12
50Feat-10CTC9	25.46	33.50	41.61	24.27	24.43	28.31	1322.43	26.65
50Feat-10CTC10	20.60	25.76	35.95	18.73	19.27	23.64	985.06	21.23
100Feat-10CTC1	382.07	474.57	399.82	387.02	399.81	420.55	-	392.96
100Feat-10CTC2	308.98	371.09	319.09	305.58	306.55	331.85	-	308.45
100Feat-10CTC3	253.20	315.33	267.57	254.31	253.82	263.03	-	254.82
100Feat-10CTC4	253.92	332.44	261.86	245.46	248.16	273.64	-	249.62
100Feat-10CTC5	208.64	278.37	213.10	199.27	202.36	220.61	-	204.19
100Feat-10CTC6	207.90	260.39	217.88	202.09	202.13	222.91	-	201.49
100Feat-10CTC7	167.58	215.10	177.24	161.59	162.89	176.21	-	164.30
100Feat-10CTC8	450.22	528.68	481.20	461.59	451.61	475.61	-	454.84
100Feat-10CTC9	258.81	333.72	269.24	250.77	250.41	279.87	-	253.23
100Fea-10CTC10	236.02	297.37	244.35	227.10	234.61	247.14	-	231.39
Apache	0.98	1.14	16.99	0.68	0.86	1.14	1.86	4.71
argo-uml-spl	1.07	1.27	17.23	0.74	0.95	1.15	3.80	4.65
BerkeleyDB	34.52	52.33	43.20	33.62	34.10	33.97	22945.38	38.97
BerkeleyDBFp	0.88	0.74	18.62	0.59	0.76	0.83	1.78	3.67
BerkeleyDBMem	10.26	11.74	23.98	9.76	10.09	10.03	62.35	13.23
BerkeleyDBPerf	4.90	5.91	22.05	4.32	4.65	5.59	70.06	6.85
Curl	1.79	1.99	18.55	1.46	1.67	2.00	9.58	4.44
DesktopSearcher	3.20	3.55	21.08	2.88	3.18	3.40	28.45	5.84
fame_dbms_fm	3.72	4.15	19.94	3.15	3.41	4.44	30.15	5.77
gpl	2.69	2.82	17.44	2.61	2.85	2.89	16.97	5.93
LinkedList	5.53	6.35	21.03	5.11	5.42	5.84	53.56	8.14
LLVM	1.34	1.18	19.22	0.90	1.14	1.36	4.22	3.60
PKJab	1.12	1.00	14.68	0.85	1.05	1.03	3.88	4.80
Prevayler	0.43	0.41	12.12	0.35	0.52	0.47	0.60	4.00
SensorNetwork	7.15	8.73	19.69	6.99	7.15	7.39	108.84	10.97
TankWar	16.41	20.51	29.99	14.64	15.26	16.76	400.78	17.88
Wget	3.43	4.37	17.15	3.16	3.44	3.92	23.37	6.31
x264	4.49	5.32	17.22	4.14	4.48	4.74	27.00	7.98
ZipMe	0.67	0.56	14.68	0.48	0.65	0.63	1.05	3.84
DrupalFM	29.84	29.34	46.81	25.85	27.47	31.10	2261.00	29.29

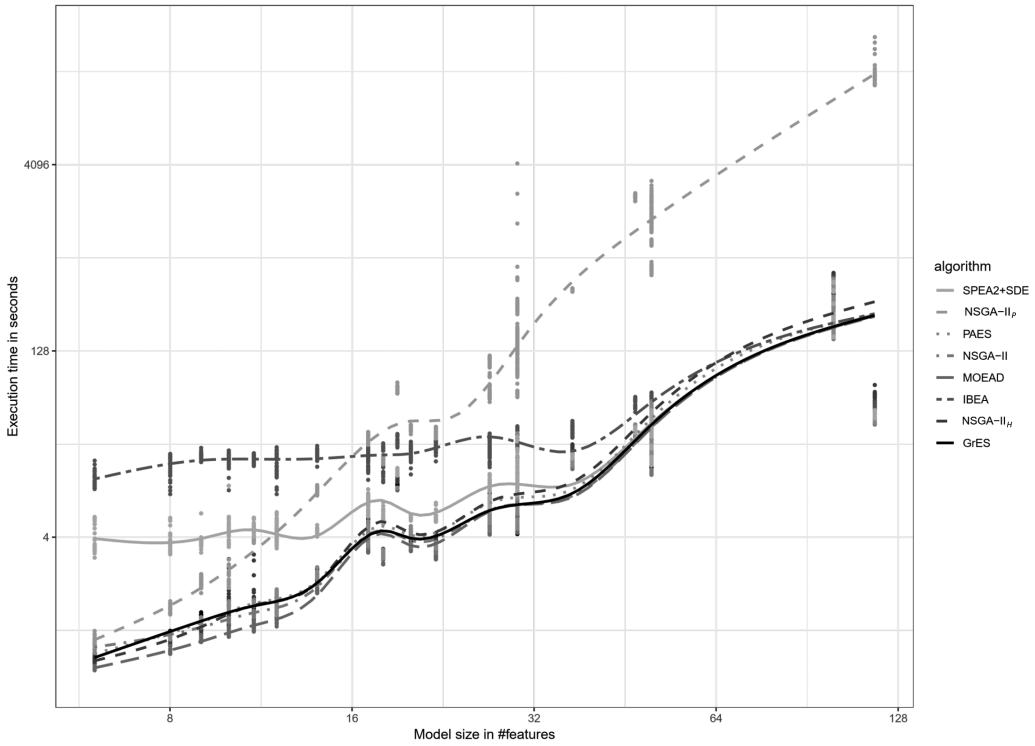

Fig. 4. Average execution times (in seconds) per algorithm and model size (logarithmic scale) using nine objectives.

of constraints, and types of CTCs, points tend to cluster in columns. In order to aid the visualisation of the execution time, an interpolation curve[4] per algorithm has been added to Figure 4.

From this we can see that GrES is consistently slightly faster than NSGA-II$_H$ and always much faster than NSGA-II$_P$. The other algorithms, that all first optimise on pairwise coverage, have similar execution time. In particular, this execution time appears to converge as the feature model size increases.

Figure 4 clearly shows that NSGA-II$_P$ is slowest for models with more than 16 features, followed by IBEA. One of the possible causes for such relatively high execution time could be the *JAVA* implementation of NSGA-II$_P$, while the other algorithms are implemented in *C++*. Moreover, IBEA was slowest for models with fewer than 16 features. For larger models, the remaining algorithms provide similar results with GrES providing the shortest time in many cases.

In order to delve deeper on the scalability of GrES, we executed an additional experiment using 5 different features models with 500 features and a 10% of CTC. The execution times of GrES for such big models were significantly longer, with a minimum of 6.9 and a maximum of 13.5 hours, having an average execution time of 9 hours. The main factor for such differences in models with the same size is the shape and structure of constraints of the specific feature models, with a small variability of execution time for the executions on the same feature model. Figure 5 depicts the scalability of GrES for all the models with model size (in number of features) in the horizontal

[4]The curve was computed using a generalized additive model which was estimated by a quadratically penalised likelihood approach [74].

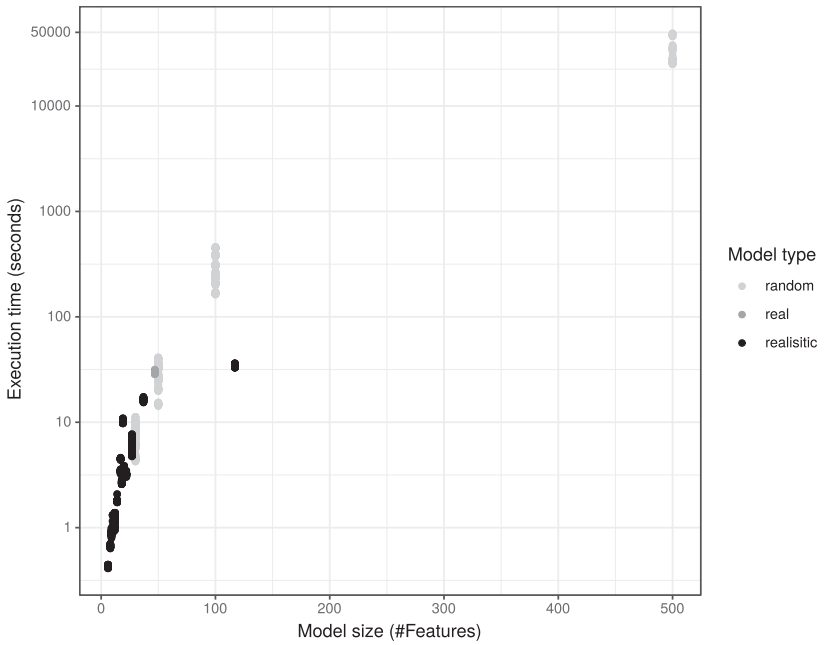

Fig. 5. Execution times (in seconds, using logarithmic scale in the vertical axis) of each GrES execution per model size using nine objectives.

axis, and execution time (measured in seconds) in the vertical axis using a logarithmic scale. Each point represents a run of the algorithm, and the specific shade of grey is used to represent the type of model used (real, realistic, or randomly generated). All the executions were performed without problems for such large models obtaining results in less than 14 hours.

5.6 Discussion

In this section, we review what the results of the experiments tell us about the research questions. First consider Research Question 1, which concerns how the performance (i.e., FCS and HV) of GrES compares with two related techniques (NSGA-II$_H$ and NSGA-II$_P$). Encouragingly, GrES always returned a population in which all solutions had full pairwise-coverage. Thus, the proposed approach provided the Software Engineer with many alternative solutions representing different trade-offs between objectives. In contrast, N SGA-II$_H$and N SGA-II$_P$always returned an FCS (the proportion of solutions in the final population that had full pairwise coverage) of below 10%. In addition, GrES also outperformed both of the previously reported approaches when we consider the HV values. With the randomly generated models, this difference appears to increase as the model size increases. Similar results were obtained with the realistic models including the feature model (Drupal) for which we had real attribute values. Finally, PLEDGE typically did not produce test suites with full pairwise coverage and in most cases GrES produced better values for the objective functions.

The second research question concerned the effectiveness (i.e., higher prioritisation values and lower costs) of the proposed evolution strategy. Again the results were promising: when we used our approach to comparing individuals, we found that the two evolution strategies used (GrES and PAES) produced populations with higher HV values than the four multi-objective optimisation algorithms considered (all had 100% FCR). This confirms our decision to use an evolution strategy.

Similar to the first set of experiments, the improvement seems to be greater for larger models. In addition, GrES led to higher HV values than PAES for the majority of the models, particularly on larger models. This verifies the effectiveness of the problem-based selection operation in GrES which can produce more promising offspring on complex problems.

We noted that previous work had used fewer objective functions and the third research question concerned how GrES performed, in terms of HV values, when compared with the previous algorithms using the (smaller number of) objective functions reported in the corresponding papers. Note that the previously reported approaches optimised on these smaller sets of objective functions but we compared them (on these objective functions) with results returned using GrES optimising on all of the objective functions. Unsurprisingly, the results of these experiments were more mixed. In these experiments, GrES typically led to lower HV values than NSGA-II$_H$ and NSGA-II$_P$ when we used the smaller models (with 30 features). However, GrES tended to return populations with higher HV values than NSGA-II$_H$ and NSGA-II$_P$ with larger models. Encouragingly, many of the differences in favour of GrES had large effect-size. In addition, in those cases where another technique returned higher HV values than GrES, the effect size was small.

The following are the key outcomes of the experiments.

(1) PLEDGE finds solutions with relatively high pairwise coverage but in almost all cases this coverage was not 100%. It may thus be unsuitable in cases where full pairwise coverage is required.
(2) The variants of NSGA-II were found to perform well, in terms of FCR and HV, for problems with 2 objectives and small feature models (i.e., around 30 features).
(3) The variants of NSGA-II also performed well, in terms of FCR and HV, for problems with up to 6 objectives and small feature models (approximately 30 features) where execution time is not an issue.
(4) GrES was found to be the most effective, in terms of FCR and HV, for problems with more than 2 objectives, and medium or large feature models (more than 30 features) and time constraints. It was much more effective than the variants of NSGA-II and PLEDGE, in terms of FCR.
(5) Both GrES and PAES were effective in finding solutions with complete pairwise coverage. However, for larger models GrES tended to produce higher HV values (in 9 out of 10 experiments, with the differences being statistically significant) and so a more diverse set of solutions.

Overall, the results were very promising. In the experiments, GrES outperformed the previously published techniques in terms of both FCR and HV. It also tended to lead to higher HV values than the other multi-objective optimisation algorithms. It is worth noting that while PAES was also highly effective when we used our novel strategy (first optimise on pairwise coverage), the original PAES does not scale well to problems with many objectives.[5] In our experiments, we adapted PAES (by using the individual-centred calculation) to make it workable in many-objective optimisation. Even so, it was still slightly slower than GrES.

The proposed approach provides Software Engineers with a wide range of objective functions to choose from. In addition, GrES is able to optimise these objective functions simultaneously while prioritising pairwise coverage, returning a diverse set of solutions that represent alternative trade-offs between the objectives. This makes the approach highly flexible and suitable for different SPL scenarios where optimisation goals may differ, e.g., testing a cyber-physical SPL vs testing a web SPL.

[5]For the considered settings, the original PAES needs to examine 10^8 grids during every generation.

6 THREATS TO VALIDITY

This section briefly discusses threats to validity and how these were addressed. We consider three types of threats: those to internal validity, construct validity, and external validity.

Internal Validity. This concerns any factors that could lead to bias. Threats could arise from the tools used in the experiments and so, where possible, we used tools that have been used in previous experiments (e.g., CASA and PLEDGE). We carefully tested our implementation of GrES and the code of the other EMO algorithms was taken from their authors except that of NSGA-II$_H$, which was written by ourselves. In addition, all experiments were run 30 times to reduce the effect of the stochastic nature of the techniques and standard statistical tests were used. HV values were computed using sampling with 10,000,000 values. The choice of reference point may introduce bias and so we followed common practice of basing this on an estimate of the Pareto front.

All of the EMO algorithms have configuration parameters, such as crossover and mutation rate, that could affect performance (i.e., higher prioritisation values and lower costs). We chose to use recommended values found in the literature, rather than carry out parameter tuning. Naturally, however, parameter tuning might further improve performance. The fact that GrES works well without tuning is promising.

Finally, programs were run in two different computers, which could make execution times incomparable. We remark, however, that all seven C++ programs were run in the same computer and so the time comparison among them is fair. As detailed in Section 4.4, the JAVA program (NSGA-II$_P$) was developed and executed by a different team in another computer. It is worth mentioning, however, that this computer was significantly more powerful than the one used for the C++ programs.

Construct Validity. This concerns whether the measurements reflect properties that are of interest in practice. The objectives used in the experiments are those reported in previous studies but the Software Engineer could choose to use a subset of these. Pairwise coverage is a widely used test objective in SPL testing and so we reported the proportion of the solutions returned that achieved 100% coverage. In addition, we used HV since it is the most widely used measure of the diversity and quality of the set of solutions returned; a set with high HV provides a diverse set of good solutions. The measurements of time may well have been affected by the fact that one algorithm was implemented in JAVA and the others in C++.

External Validity. This refers to the degree to which we can generalise from the experiments. This threat will always exist in Software Engineering research since we do not know the population of real problems and have no way of sampling from this in a uniform manner. We reduced this threat by using a mixture of randomly generated feature models, 19 realistic feature models with randomly generated attribute values, and 1 realistic feature model with real attribute values.

7 RELATED WORK

Recent surveys and mapping studies on SPL testing reveal an increasing interest in the topic [15, 20, 21, 49]. In this section, we briefly summarise those papers that address the problem of test case selection and prioritisation from feature models using both multi and single-objective approaches.

Multi-objective testing for SPL. Wang et al. [73] presented an industrial case study on multi-objective test case prioritisation for SPL. The problem was modelled using a single fitness function with four weighted objectives: minimising execution cost, maximising number of prioritised test cases, maximising pairwise coverage, and maximising fault detection capability. Four search-based algorithms were compared, namely, alternative variable method, a custom genetic algorithm, (1+1)

evolutionary algorithm, and random search. The experiments were conducted on a real feature model with 53 features and 500 randomly generated models with up to 500 features.

Henard et al. [31] presented an ad-hoc multi-objective genetic algorithm for test case selection from feature models. Three objectives were combined into a single fitness function with weighting factors: maximising pairwise coverage, minimising number of test cases, and minimising test cost. Their approach was compared with random solutions for 8 feature models from the SPLOT repository with up to 94 features.

Lopez-Herrejon et al. [47] proposed an approach for computing the exact Pareto front of a feature model using SAT solvers. Two objectives were optimised: maximising pairwise coverage and minimising test suite size. As expected, the approach suffers from scalability issues: the Pareto front could be calculated in only 10 out of the 20 realistic feature models used in this study. In a subsequent work [48], the authors compared four classical multi-objective algorithms (NSGA-II, MOCell, SPEA2, and PAES) and the impact of three different seeding strategies on computing test suites with maximum coverage and minimum size. Their approach was evaluated on 19 out of the 20 feature models presented in Table 5 (the Drupal feature model was presented later). In a later paper [49], the authors presented an overview of the state of the art on evolutionary computation for SPL testing. As a part of their work, they identified some open challenges including the application of many-objectives algorithms and the use of non-functional properties for test case prioritisation.

Wang et al. [70] presented an approach to minimise SPL test suites by eliminating redundant test cases, where test cases are inputs and expected outputs to test products (rather than products). Given an input product and a test suite for testing it, their approach searches for a "good" subset of the suite according to several effectiveness and cost-related objectives, i.e. a separate optimisation problem is solved for each product. Their work compared three genetic algorithms where three objectives were combined into a single fitness function with weighting factors: minimising the number of test cases, maximising pairwise coverage, and maximising fault detection capability. Their approach was evaluated using several models from the SPLOT repository plus four feature models from an industrial project with sizes ranging from 17 to 77 features. In subsequent work [71], the authors compared their weight-based genetic algorithms against seven multi-objective search-based algorithms, namely NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE and random search. Four fitness functions related to test effectiveness (to be maximised) and one fitness function to measure cost (to be minimised) were presented. The experiments were conducted on four feature models from an industrial project and 500 random feature models with up to 1,000 features. Algorithms were compared using a custom metric that combines the values of the five fitness functions for each solution.

Lopez-Herrejon et al. [50] presented a genetic algorithm for the generation of prioritised pairwise test suites of minimum size from a feature model. The algorithm follows a master-slave strategy to parallelise the evaluation of products, which are prioritised based on some user-defined weights. In a related work [24], some of the authors presented two hybrid algorithms based on integer programming to address the same problem, outperforming the parallel genetic algorithm in both solution quality and computation time.

Devroey et al. [19] proposed a search-based approach for test case selection in SPLs maximising the distance between products (in terms of features) and the distance between behavioural actions, derived from a model representing the behaviour of the SPL named Featured Transition System. Both distances were combined into a single fitness function integrated into the (1+1) Evolutionary Algorithm. Their approach was evaluated using four feature models with up to 44 features.

Ferreira et al. [23, 65] presented a hyper-heuristic approach for dynamic selection of evolutionary operators to be applied during the execution of a multi-objective evolutionary algorithm. The

hyper-heuristic tries to determine the best mutation and crossover operators based on their performance. In their paper, they compared several hyper-heuristics on NSGA-II, SPEA2, IBEA, and MOEA/D-DRA with four objective functions: minimising suite size, maximising mutation score, maximising pairwise coverage, and maximising dissimilarity of products. The approach was evaluated using four realistic feature models with sizes ranging from 14 to 22 features.

Parejo et al. [57] presented a case study on multi-objective test case prioritisation in highly-configurable s ystems u sing t he D rupal W eb f ramework [59]. T hey p roposed s even prioritisation objective functions based on functional information extracted from the feature model, and non-functional data extracted from code and issue repositories. The approach was evaluated by comparing the effectiveness of 63 different combinations of up to three objective functions at accelerating the detection of faults in Drupal using the NSGA-II algorithm.

Markiegi et al. [54] proposed a genetic algorithm for test selection using three objectives combined into a single fitness function with weighting factors: maximising fault detection capability, minimising test execution time, and maximising test case appearance frequency. The approach was evaluated using a cyber-physical SPL with 46 features.

Our work is related to articles on multi-objective product selection where the goal is to select a product that optimises two or more objectives (see, for example, [26], [27], [33], [34], [61]). Compared to them, we address a different problem—the selection of a sequence of products (a test suite)—which means that we could not directly apply the algorithms used for optimal product selection. This being said, we see some potential in applying some of the ideas used in the generation of optimal products in our future work, such as enhancing the diversity of randomly generated products [26, 33].

Single-Objective Testing for SPL. A vast majority of the single-objective test case selection techniques for SPL are based on feature coverage using combinatorial techniques. Lopez-Herrejon et al. conducted a systematic mapping study on combinatorial interaction testing for SPLs [52]. They identified over forty approaches using different techniques such as genetic and greedy algorithms. They also found that a majority of the papers focus on deriving products from variability models (typically a feature model) using pairwise testing.

Henard et al. [32] proposed a mutation-based approach for test case selection in SPLs that works in two steps. First, a set of faulty versions of the feature model, so-called mutants, are created, e.g., removing an excludes constraint. Then, the (1+1) Evolutionary Algorithm is used to search for test cases that kill as many mutants as possible, i.e., products that do not conform to the mutated feature models.

Henard et al. [29] presented a search-based approach to generate and prioritise t-wise test suites, with $t \in \{2, \ldots, 6\}$, on some of the largest features model available (with up to 6.8K features). They employ a variant of the (1+1) Evolutionary Algorithm with no crossover and a fitness function based on the Jaccard's dissimilarity metric. The initial population is generated randomly using a modified SAT solver to guarantee that solutions are generated in an unpredictable way. Their work is based on the assumption that the higher the differences between the products of the suite, the higher the t-wise coverage. The suite is also prioritised based on the dissimilarity distances among the products in a subsequent step. Their algorithm computes suites of fixed size within a given time used as a stopping criterion. In related work, Al-Hajjaji et al. [3] proposed an algorithm for SPL prioritisation using an adaptation of the Hamming distance to measure the similarity among products.

Xu et al. [75] proposed an SPL test suite augmentation approach and related tool named CONTESA. Their tool incrementally generates test cases that exercise parts of the code that have not yet been covered by previous test cases, e.g., code branches. A genetic algorithm was used to

automatically generate test cases for the products under test. Devroy et al. [18] proposed to prioritise products according to the likelihood of their executions, derived from a usage model represented by a Featured Transition System (a type of Markov chain).

Ensan et al. [22] proposed a genetic algorithm approach for the generation of SPL test suites with good fault detection capability and feature coverage. The fitness of each suite was measured using a combination of the variability coverage and cyclomatic complexity metrics for feature models [7]. Inspired by their work, those metrics were adapted and used in the `VCoverage` prioritisation function used in this paper. In related work, the authors proposed eight coverage criteria based on the transformation of feature models into formal context-free grammars [6]. The rationale behind the proposed coverage criteria is based on the development of equivalence partitions on the SPL under test and the use of boundary–value analysis for test case generation.

Al-Hajjaji et al. [1] proposed a cluster-based approach for similarity-based product prioritisation. The approach was evaluated using 10 feature models with up to 6,888 features. In a related paper [2], the authors proposed a prioritisation approach based on product similarity using delta-modeling, that is, specific information about how each product is implemented. The approach was evaluated by means of an SPL from the automotive domain with 27 features.

Finally, the synergies between SBSE and SPLs have been extensively explored and summarised in several survey papers [28, 53]. This article takes a step further in exploiting the benefits of many-objective search-based optimisation for testing SPLs.

Compared to the extensive body of related research, this article presents several novel contributions. First, we propose the first many-objective approach for SPL testing optimising up to 9 different objectives, addressing both test selection and prioritisation simultaneously. Also, we present a novel evolutionary algorithm (GrES) and an optimisation framework, where pairwise coverage is given priority over the rest of the objectives. The evaluation results with both random and realistic feature models show that this approach is able to tame the complexity of the problem and outperform the state-of-the-art algorithms used in previous approaches.

8 CONCLUSIONS

This paper investigated the problem of producing good test suites for SPL testing: test suites that achieve full pairwise coverage and also have a number of other desirable properties related to test selection and test prioritisation. Test generation is thus a many-objective optimisation problem, with different solutions (that are not related under Pareto dominance) representing alternative trade-offs that can be provided to the Software Engineer.

We noted that one of the objectives, pairwise coverage, is more important than the others: the Software Engineer is only interested in tests that achieve full coverage. As a result, we introduced a novel approach, based on this piece of domain knowledge, that optimises first on pairwise coverage and only then on the other objectives. Importantly, it is straightforward to adapt any EMO to use this approach. We then developed a novel evolution strategy, GrES, that uses this approach. GrES adapts a standard evolution strategy to reflect domain knowledge including the fact that the set of objectives can be partitioned into two sets (for selection and prioritisation). We had observed that the crossover operation could break good building blocks of chromosomes of the seed individual in the evolutionary search, and thus GrES conducts repeated explorations around the seed individual and its offspring.

We evaluated the proposed approach, GrES, through a range of experiments that addressed the following research questions: whether GrES outperformed current techniques (in terms of both FCR, the number of test suites with full pairwise coverage, and the HV values of the populations returned); whether the novel evolution strategy used in GrES contributed to its performance; how GrES performed with the subsets of objectives used in previous studies; whether differences in

performance were substantial; and whether different techniques had different execution times. The experiments were carried out on randomly generated feature models, allowing us to explore how performance varies with feature model size, and also on realistic feature models. We had 20 realistic feature models and used randomly generated attribute values for 19 of these and real attribute values for one. The results were generally very positive. In the experiments, GrES always returned a population in which all solutions provided full pairwise coverage. In contrast, in all cases the previously published approaches returned populations in which less than 10% of the solutions had full pairwise coverage (averaged over 30 runs). In addition, GrES returned populations with higher HV values. When we used the approach (optimise first on pairwise coverage) with other EMOs we found that all approaches produced populations in which all test suites provided full pairwise coverage but that the evolution strategies (GrES and PAES) led to higher HV values than the other EMOs, especially with the larger models. GrES appeared to perform a little better than PAES, in terms of HV values, but the results were mixed. However, GrES has much lower computational requirements than PAES, requiring us to adapt PAES (otherwise it could not be used in many-objective problems). This is not surprising since PAES examines an exponential number of grids.

There are a number of possible lines of future work. First, we could fix the time given to the techniques rather than the number of evaluations. Second, parameter tuning may well lead to further improvements in performance. In addition, there is the problem of finding techniques that scale to larger feature models and the potential to incorporate approaches for enhancing the diversity of random products (as already done in the selection of optimal products [26, 33]). Finally, there is value in repeating the experiments with other realistic feature models.

MATERIALS

For the sake of verifiability, our implementations of the algorithms in Java and C++, all the raw data generated during its execution, as well as all models and artefacts of the experiments are available at https://drive.google.com/open?id=1xumU6qxBesloq69jOPMbprOaiaOKDq82.

REFERENCES

[1] M. Al-Hajjaji, J. Krüger, S. Schulze, T. Leich, and G. Saake. 2017. Efficient product-line testing using cluster-based product prioritization. In *Proceedings of the 2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*. 16–22. DOI: https://doi.org/10.1109/AST.2017.7

[2] M. Al-Hajjaji, S. Lity, R. Lachmann, T. Thüm, I. Schaefer, and G. Saake. 2017. Delta-oriented product prioritization for similarity-based product-line testing. In *Proceedings of the 2nd International Workshop on Variability and Complexity in Software Design (VACE'17)*. IEEE Press, Piscataway, N.J. 34–40. DOI: https://doi.org/10.1109/VACE.2017..8

[3] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake. 2016. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling* (2016), 1–23. DOI: https://doi.org/10.1007/s10270-016-0569-2

[4] T. Back. 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press.

[5] J. Bader and E. Zitzler. 2011. HypE: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary Computation* 19, 1 (2011), 45–76.

[6] E. Bagheri, F. Ensan, and D. Gasevic. 2012. Grammar-based test generation for software product line feature models. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'12)*. IBM Corp., Riverton, N.J. 87–101. http://dl.acm.org/citation.cfm?id=2399776.2399785.

[7] E. Bagheri and D. Gasevic. 2011. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal* 19, 3 (2011), 579–612. DOI: https://doi.org/10.1007/s11219-010-9127-2

[8] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski. 2016. Clafer: Unifying class and feature modeling. *Software & Systems Modeling* 15, 3 (1 Jul 2016), 811–845. DOI: https://doi.org/10.1007/s10270-014-0441-1

[9] D. Benavides, S. Segura, and A. A. Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636. DOI: https://doi.org/10.1016/j.is.2010.01.001

[10] K. Bringmann and T. Friedrich. 2010. The maximum hypervolume set yields near-optimal approximation. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*. ACM press, 511–518.

[11] P. Clements and L. Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison–Wesley.

[12] C. A. Coello, D. A. Van Veldhuizen, and G. B. Lamont. 2007. *Evolutionary Algorithms for Solving Multi-Objective Problems* (2 ed.). Springer, Munich.

[13] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates. 2001. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Genetic and Evolutionary Computation Conference (GECCO'01)*. 283–290.

[14] D. W. Corne and J. D. Knowles. 2007. Techniques for highly multiobjective optimisation: Some nondominated points are better than others. In *Proceedings of the 9th Annual Genetic and Evolutionary Computation Conference (GECCO)*. 773–780.

[15] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. Santana de Almeida, and S. Romero de Lemos Meira. 2011. A systematic mapping study of software product lines testing. *Information & Software Technology* 53, 5 (2011), 407–423.

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

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[18] X. Devroey, G. Perrouin, M. Cordy, P.-Y. Schobbens, A. Legay, and P. Heymans. 2013. Towards statistical prioritization for software product lines testing. In *Proceedings of the 8th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'14)*. ACM, New York, Article 10, 7 pages. DOI: https://doi.org/10.1145/2556624.2556635

[19] X. Devroey, G. Perrouin, A. Legay, P.-Y. Schobbens, and P. Heymans. 2016. Search-based similarity-driven behavioural SPL testing. In *Proceedings of the 10th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'16)*. ACM, New York, 89–96. DOI: https://doi.org/10.1145/2866614.2866627

[20] I. do Carmo Machado, J. D. McGregor, Y. Cerqueira Cavalcanti, and E. Santana de Almeida. 2014. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology* 56, 10 (2014), 1183–1199. DOI: https://doi.org/10.1016/j.infsof.2014.04.002

[21] E. Engström and P. Runeson. 2011. Software product line testing - A systematic mapping study. *Information and Software Technology* 53, 1 (2011), 2–13.

[22] F. Ensan, E. Bagheri, and D. Gašević. 2012. Evolutionary search-based test generation for software product line feature models. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering. (CAiSE 2012), Gdansk, Poland, June 25–29, 2012. Proceedings*. Springer, Berlin, 613–628. DOI: https://doi.org/10.1007/978-3-642-31095-9_40

[23] T. N. Ferreira, J. A. P. Lima, A. Strickler, J. N. Kuk, S. R. Vergilio, and A. Pozo. 2017. Hyper-heuristic based product selection for software product line testing. *IEEE Computational Intelligence Magazine* 12, 2 (May 2017), 34–45. DOI: https://doi.org/10.1109/MCI.2017.2670461

[24] J. Ferrer, F. Chicano, and E. Alba. 2017. Hybrid algorithms based on integer programming for the search of prioritized test data in software product lines. In *Proceedings of the 20th European Conference on Applications of Evolutionary Computation Part II, (EvoApplications 2017), (Amsterdam, The Netherlands, April 19–21, 2017)*. Giovanni Squillero and Kevin Sim (Eds.). Springer International Publishing, Cham, 3–19. DOI: https://doi.org/10.1007/978-3-319-55792-2_1

[25] B. J. Garvin, M. B. Cohen, and M. B. Dwyer. 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering* 16, 1 (1 Feb 2011), 61–102. DOI: https://doi.org/10.1007/s10664-010-9135-7

[26] J. Guo, Jia H. Liang, K. Shi, D. Yang, J. Zhang, K. Czarnecki, V. Ganesh, and H. Yu. 2019. SMTIBEA: A hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Software & Systems Modeling* 18, 2 (1 Apr 2019), 1447–1466. DOI: https://doi.org/10.1007/s10270-017-0610-0

[27] J. Guo, E. Zulkoski, R. Olaechea, D. Rayside, K. Czarnecki, S. Apel, and J. M. Atlee. 2014. Scaling exact multi-objective combinatorial optimization by parallelization. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*. ACM, New York, 409–420. DOI: https://doi.org/10.1145/2642937.2642971

[28] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. 2014. Search based software engineering for software product line engineering: A survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC'14)*. ACM, New York, 5–18. DOI: https://doi.org/10.1145/2648511.2648513

[29] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon. 2014. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering* 40 (2014), 1.

[30] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. 2013. PLEDGE: A Product Line Editor and Test Generation Tool. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops (SPLC'13 Workshops)*. ACM, New York, 126–129. DOI: https://doi.org/10.1145/2499777.2499778

[31] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. 2013. Multi-objective test generation for software product lines. In *International Software Product Line Conference (SPLC)*.

[32] C. Henard, M. Papadakis, and Y. Le Traon. 2014. Mutation-based generation of software product line test configurations. In *Proceedings of the6th International Symposium on Search-Based Software Engineering: , (SSBSE 2014), (Fortaleza, Brazil, August 26–29, 2014)*. Springer International Publishing, Cham, 92–106. DOI: https://doi.org/10.1007/978-3-319-09940-8_7

[33] C. Henard, M. Papdakis, M. Harman, and Y. Le Traon. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 2015 International Conference on Software Engineering (ICSE'15)*. IEEE Press.

[34] R. M. Hierons, M. Li, X. Liu, S. Segura, and W. Zheng. 2016. SIP: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology* 25, 2 (2016), 17.

[35] P. Hofman, T. Stenzel, T. Pohley, M. Kircher, and A. Bermann. 2012. Domain specific feature modeling for software product lines. In *Proceedings of the 16th International Software Product Line Conference - Volume 1 (SPLC'12)*. ACM, New York, 229–238. DOI: https://doi.org/10.1145/2362536.2362568

[36] M. F. Johansen, O. Haugen, and F. Fleurey. 2012. An algorithm for generating T-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference - Volume 1 (SPLC'12)*. ACM, New York, 46–55. DOI: https://doi.org/10.1145/2362536.2362547

[37] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. SEI.

[38] J. D. Knowles and D. W. Corne. 1999. The pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation. In *Proceedings of the Congress on Evolutionary Computation (CEC'99)*, Vol. 1.

[39] J. D. Knowles and D. W. Corne. 2000. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation* 8, 2 (June 2000), 149–172.

[40] B. Li, J. Li, K. Tang, and X. Yao. 2015. Many-objective evolutionary algorithms: A survey. *Computing Surveys* 48, 1 (2015), 1–35.

[41] K. Li, R. Wang, T. Zhang, and H. Ishibuchi. 2018. Evolutionary many-objective optimization: A comparative study of the state-of-the-art. *IEEE Access* 6 (2018), 26194–26214.

[42] M. Li, T. Chen, and X. Yao. 2018. A critical review of "A Practical Guide to Select Quality Indicators for Assessing Pareto-Based Search Algorithms in Search-Based Software Engineering": Essay on quality indicator selection for SBSE. In *Proceedings of the 40th International Conference on Software Engineering (ICSE): New Ideas and Emerging Results Track*. 17–20.

[43] M. Li, C. Grosan, S. Yang, X. Liu, and X. Yao. 2018. Multi-line distance minimization: A visualized many-objective test problem suite. *IEEE Transactions on Evolutionary Computation* 22, 1 (2018), 61–78.

[44] M. Li, S. Yang, and X. Liu. 2014. Shift-based density estimation for Pareto-based algorithms in many-objective optimization. *IEEE Transactions on Evolutionary Computation* 18, 3 (2014), 348–365.

[45] M. Li and X. Yao. 2019. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 26.

[46] M. Li, L. Zhen, and X. Yao. 2017. How to read many-objective solution sets in parallel coordinates [Educational Forum]. *IEEE Computational Intelligence Magazine* 12, 4 (2017), 88–100.

[47] R. E. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, and E. Alba. 2013. Multi-objective optimal test suite computation for software product line pairwise testing. In *Proceedings of the 29th IEEE International Conference on Software Maintenance*.

[48] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba. 2014. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6–11, 2014*. 387–396. DOI: https://doi.org/10.1109/CEC.2014.6900473

[49] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba. 2016. Evolutionary computation for software product line testing: An overview and open challenges *Computational Intelligence and Quantitative Software Engineering*. Springer International Publishing, Cham, 59–87. DOI: https://doi.org/10.1007/978-3-319-25964-2_4

[50] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. 2014. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO'14)*. ACM, New York, 1255–1262. DOI: https://doi.org/10.1145/2576768.2598305

[51] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. 2014. Towards a benchmark and a comparison framework for combinatorial interaction testing of software product lines. *CoRR* abs/1401.5367 (2014). http://arxiv.org/abs/1401.5367.

[52] R. E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed. 2015. A first systematic mapping study on combinatorial interaction testing for software product lines. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation, (ICST 2015 Workshops), (Graz, Austria, April 13–17, 2015)*. 1–10. DOI: https://doi.org/10.1109/ICSTW.2015.7107435

[53] R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed. 2015. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology* 61 (2015), 33–51. DOI: https://doi.org/10.1016/j.infsof.2015.01.008

[54] U. Markiegi, A. Arrieta, G. Sagardui, and L. Etxeberria. 2017. Search-based product line fault detection allocating test cases iteratively. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A (SPLC'17)*. ACM, New York, 123–132. DOI: https://doi.org/10.1145/3106195.3106210

[55] Y. Matsumoto. 2007. A guide for management and financial controls of product lines. In *Proceedings of the 11th International Software Product Line Conference*. 162–170.

[56] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki. 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC'14)*. ACM, New York, 92–101. DOI: https://doi.org/10.1145/2648511.2648521

[57] J. A. Parejo, A. B. Sánchez, S. Segura, A. Ruiz-Cortés, R. E. Lopez-Herrejon, and A. Egyed. 2016. Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software* 122 (2016), 287–310.

[58] M. Ravber, M. Mernik, and M. Črepinšek. 2017. The impact of quality indicators on the rating of multi-objective evolutionary algorithms. *Applied Soft Computing* 55 (2017), 265–275.

[59] A. B. Sánchez, S. Segura, J. A. Parejo, and A. Ruiz-Cortés. 2015. Variability testing in the wild: The Drupal case study. *Software and Systems Modeling* (2015), 1–22. DOI: https://doi.org/10.1007/s10270-015-0459-z

[60] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. 2014. A comparison of test case prioritization criteria for software product lines. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation (ICST'14)*. IEEE Computer Society, Washington, DC, 41–50. DOI: https://doi.org/10.1109/ICST.2014.15

[61] A. S. Sayyad, T. Menzies, and H. Ammar. 2013. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*. IEEE Press, Piscataway, N.J. 492–501. http://dl.acm.org/citation.cfm?id=2486788.2486853.

[62] H.-P. Schwefel. 1993. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc.

[63] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés. 2012. BeTTy: Benchmarking and testing on the automated analysis of feature models. In *Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12)*. ACM, New York, 63–71. DOI: https://doi.org/10.1145/2110147.2110155

[64] D. C. Sharp. 1998. Reducing avionics software cost through component based product line development. In *Proceedings of the 17th Digital Avionics Systems Conference*. IEEE.

[65] A. Strickler, J. A. Prado Lima, S. R. Vergilio, and A. Pozo. 2016. Deriving products for variability test of feature models with a hyper-heuristic approach. *Applied Soft Computing* 49 (2016), 1232–1242. DOI: https://doi.org/10.1016/j.asoc.2016.07.059

[66] T. Thüm, D. Batory, and C. Kastner. 2009. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*. IEEE Computer Society, Washington, DC, 254–264. DOI: https://doi.org/10.1109/ICSE.2009.5070526

[67] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70–85. DOI: https://doi.org/10.1016/j.scico.2012.06.002 Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).

[68] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. 2008. FAMA framework. In *Proceedings of the 12th Software Product Lines Conference (SPLC)*. 359. DOI: https://doi.org/10.1109/SPLC.2008.50

[69] A. Vargha and H. D. Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[70] S. Wang, S. Ali, and A. Gotlieb. 2013. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*.

[71] S. Wang, S. Ali, and A. Gotlieb. 2015. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software* 103 (2015), 370–391. DOI: https://doi.org/10.1016/j.jss.2014.08.024

[72] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen. 2016. A practical guide to select quality indicators for assessing Pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. 631–642.

[73] S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, and M. Liaaen. 2014. Multi-objective test prioritization in software product line testing: An industrial case study. In *Software Product Line Conference*. 32–41.

[74] S. N. Wood, N. Pya, and B. Säfken. 2016. Smoothing parameter and model selection for general smooth models. *J. Amer. Statist. Assoc.* 111, 516 (2016), 1548–1563. DOI: https://doi.org/10.1080/01621459.2016.1180986

[75] Z. Xu, M. B. Cohen, W. Motycka, and G. Rothermel. 2013. Continuous test suite augmentation in software product lines. In *Proceedings of the 17th International Software Product Line Conference (SPLC'13)*. ACM, New York, 52–61. DOI: https://doi.org/10.1145/2491627.2491650

[76] S. Yang, M. Li, X. Liu, and J. Zheng. 2013. A grid-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation* 17, 5 (2013), 721–736.

[77] S. Yoo and M. Harman. 2012. Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability* 22, 2 (2012), 67–120. DOI: https://doi.org/10.1002/stvr.430

[78] Q. Zhang and H. Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007), 712–731.

[79] A. Zhou, B. Y. Qu, H. Li, S. Z. Zhao, P. N. Suganthan, and Q. Zhang. 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation* 1, 1 (2011), 32–49.

[80] E. Zitzler and S. Künzli. 2004. Indicator-based selection in multiobjective search. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN)*. 832–842.

[81] E. Zitzler and L. Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (1999), 257–271.