Looking For Novelty in Search-based Software Product Line Testing

Xiang Yi [1]

[1]South China University of Technology

October 30, 2023

Abstract

This paper show how the novelty search (NS) algorithm can be used for similarity-based software product line testing

Looking For Novelty in Search-based Software Product Line Testing

Yi Xiang, Han Huang *Senior Member, IEEE,* Miqing Li, Sizhe Li, and Xiaowei Yang

Abstract—Testing software product lines (SPLs) is difficult due to a huge number of possible products to be tested. Recently, there has been a growing interest in similarity-based testing of SPLs, where similarity is used as a surrogate metric for the t-wise coverage. In this context, one of the primary goals is to sample, by using search-based algorithms, a subset of test cases (i.e., products) as dissimilar as possible, thus potentially making more t-wise combinations covered. Prior works have shown, by means of empirical studies, the great potential of current similarity-based testing approaches. However, the rationale of this testing technique deserves a more rigorous exploration. To this end, we perform a correlation analysis to investigate the internal relationship between similarity metrics and the t-wise coverage. We find that similarity metrics generally have a *significantly positive correlation* with the t-wise coverage. This well explains why similarity-based testing works, as the improvement on similarity metrics will potentially increase the t-wise coverage. Moreover, we explore, for the first time, the use of the novelty search (NS) algorithm for similarity-based SPL testing. The algorithm rewards "novel" individuals, i.e., those being different from individuals discovered previously, and this perfectly matches the goal of similarity-based SPL testing. We demonstrate that the novelty score in NS has *a (much) stronger positive correlation* with the t-wise coverage than the widely used fitness function employed in the genetic algorithm (GA). Empirical results on 31 software product lines, either realistic or artificial, validate the superiority of NS over GA, as well as other state-of-the-art approaches. Finally, we investigate how the performance of NS is affected by the ways of generating new products, and by its key parameters. In summary, looking for novelty provides an alternative way of generating test cases for SPL testing.

Index Terms—Software product line testing, novelty search, similarity-based testing, t-wise coverage, correlation analysis

1 Introduction

A software product line (SPL) is a family of related products that are built from a set of features, with a feature being some aspect of the system functionality [1], and a product being a set of selected/deselected features. These products, which share some common features (i.e., *commonality*), are distinguished by specific features they provide (i.e., *variability*). The commonality and variability of an SPL are often organized by a feature model (FM) [2], [3] which defines all the possible software products by expressing relationships and constraints among features [4]. The adoption of SPLs can benefit the industry in different respects such as decreasing implementation costs, reducing time to market, and improving product quality [5]. There is evidence of numerous companies such as Bosch, Philips, Siemens, Boeing and Toshiba applying SPLs to develop softwares [6], [7], [8].

Despite the benefits that SPLs bring, it raises new challenges with respect to how to ensure the reliability of an SPL. In this regard, testing software product lines becomes crucial to avoid fault propagation to the derived products [9]. Indeed, a defect in a single feature may exist in thousands or even millions of products [7]. Nevertheless, SPL testing is an inherently difficult task because the number of possible products induced by a given FM potentially grows exponentially with the number of features. Ideally, one would like to test all of the valid products, but it is rarely possible in practice due to a huge number of possible products to be tested. Moreover, even if one could test all valid products (for small-scale FMs), it is likely to be inefficient because software testers often have a limited amount of test resources [8], like time and budgets.

Therefore, software testers are seeking sampling approaches to reduce the size of products under test so as to meet the release deadline or resource constraints [10]. The combinatorial interaction testing (CIT) [11], among others, is a prominent approach that has been introduced to reduce the size of the test suites while achieving a certain coverage of feature combinations [12], [13]. This approach systematically samples from a large domain of test data. It is based on the observation that most faults are caused by interactions among a small number of features [14]. For example, Kuhn et al. [14] have shown, using empirical data, that the interactions between two features are capable of disclosing 93% bugs for a large distributed system developed at NASA Goddard Space Flight Center. In particular, the t-wise CIT requires to find a minimal subset of products that cover all the possible interactions between t features by at least one product under test [15]. Since this is an NP-hard problem, several greedy or metaheuristic approaches have been proposed to conduct t-wise sampling, such as MoSo-PoLiTe [16], CASA [17], Chvatal [12], [18], ICPL [19], ACTS [20], IncLing [21], and those proposed in [22], [23], [24]. A review of product sampling tools for SPLs can be found in a recent survey [10].

However, existing t-wise sampling techniques face the

- *Yi Xiang, Han Huang, Sizhe Li and Xiaowei Yang are with the School of Software Engineering, South China University of Technology, Guangzhou 510006, China*
  E-mail: xiangyi@scut.edu.cn (Y. Xiang), xwyang@scut.edu.cn (X. Yang)
- *Miqing Li is with the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK.*
  E-mail: m.li.8@cs.bham.ac.uk

Manuscript received XX, 2020; revised XX, 2020.

scalability issue [25]. For large real-world SPLs, they often run out of memory, do not terminate, or take too much running time [4], [25], [26], [27], [28]. Although the t-wise testing can drastically reduce the number of products to consider (compared to testing all possible products exhaustively), this number can still be too large to fit the test budget allocated. This is particularly true if the number of features, and/or the strength t is large. For example, according to [19], 480 products are required to achieve a full 2-wise coverage for the Linux kernal FM (2.6.28.6-icse11) with 6,888 features [29]. Similarly, Song et al. [30] mentioned that one needs to test too many products to obtain a full 7-wise coverage of a realistic system. In fact, the maximum number of all the t-wise combinations of features is $C_n^t \cdot 2^t$ [14], where n is the number of features. Clearly, a huge amount of products are required to cover all the combinations in case that n and t are large. In practice, most real-world SPLs are variability-intensive, with hundreds or even thousands of features [29], like for example Linux kernal [29], Automotive02 [31] and Eclipse [32]. Moreover, there is a practical need to deal with high interaction strengths ($t > 2$) [33], [34], [35]. Petke et al. [34] showed that higher-strength test suites are able to find more faults, and Kuhn et al. [14] indicated that almost all the defects can be found for the 6-wise coverage.

As an avenue to circumvent the scalability issue faced by the t-wise CIT, the similarity testing [36], [37], [38], [39], a technique used to select a subset of test cases by increasing diversity among test cases so as to achieve a given goal, such as maximising the fault detection rate or the structural coverage criteria, was introduced to the SPL testing. Henard et al. [4] applied similarity testing to SPLs in order to sample and prioritize products to test. The key idea is to mimic t-wise testing by maximizing diversity among products. In their work, a genetic algorithm (GA) is designed to evolve a population of products by optimizing the fitness function which is defined based on a similarity metric. Their results showed that the approach is promising for large SPLs and high interaction strengths, forming a scalable and flexible alternative to the t-wise testing. Later, Fischer et al. [40] empirically assessed this approach with respect to fault detection capability in the Drupal [41] case study. They reported that Henard's similarity approach is useful to detect interaction faults in Drupal, being also able to yield a t-wise coverage comparable with CASA [17]. These findings indicate that similarity is a suitable surrogate metric for the t-wise coverage. Using a different similarity metric, Al-Hajjaji et al. [9] also adopted similarity testing for SPLs. In particular they focused on similarity-based prioritization which orders a set of the products before they are tested. In a subsequent study [7], this work was substantially extended by the same authors in the experimental evaluations using three SPLs with real faults in their source code. There have been a number of other works exploiting similarity in the context of SPL testing, e.g., [21], [42], [43], [44], [45], [46], [47], [48]. This is also the topic we focus on in this article.

The suitability of similarity as a surrogate metric for the t-wise coverage was demonstrated in the aforementioned prior works mainly by means of observing and analyzing empirical results. However, there has been no work on in-depth explanation of this, particularly from the statistics point of view. In this paper, we make a first step along this line. In particular, a correlation analysis is performed to explore the internal relationship between similarity metrics and t-wise coverage in the context of SPL testing. Our primary finding is that similarity metrics (e.g., the fitness used by Henard et al. [4]) has a *significantly positive correlation* with the t-wise coverage for most of the FMs considered in this study. This clearly suggests that an improvement on similarity metrics will potentially foster the increase of the t-wise converge, and it thus makes sense to apply a search-based method to optimize similarity metrics to indirectly increase, as much as possible, the t-wise coverage. Interpreting the rationale of similarity-based testing by using correlation analysis forms the first contribution of this paper.

Furthermore, we apply, for the first time, the novelty search (NS) algorithm [49], [50] [51] to the similarity-based testing of SPLs. Unlike classical objective-oriented evolutionary algorithm, the NS algorithm abandons objectives and it promotes evolution through the search for "novelty" in the behaviour (or decision) space, wherein "novelty" implies being different from individuals discovered previously. More precisely, instead of rewarding performance based on an objective, the NS rewards performance based on any novelty metric, which measures the uniqueness of an individual with respect to the rest of the population and an archive of past individuals [50]. This way, it creates a constant pressure to do something novel, being able to generate a set of diverse individuals. The NS has been successfully applied to a certain number of domains, e.g., maze navigation and biped locomotion [50] and the evolution of plastic neural networks [52]. It has been recently pointed out by Romero et al. [53] that NS seems to well fit precepts of search-based software engineering (SBSE). However, there has been only a very limited number of studies relating to the application of this algorithm to the SBSE field. Boussaa et al. [54] applied the NS algorithm to the test data generation problem based on statement-covered criteria. As stated by Boussaa et al. [54], that work is the first one in the literature to adopt NS to generate test data. López-López et al. [55] used, for the first time, NS for software improvement by combining it with a genetic improvement system. Recently, Romero et al. [53] have explored the use of NS for the next release problem [56]. In this paper, we apply NS to the similarity-based testing of SPLs because we notice that NS perfectly matches the goal of this testing technique, i.e., searching for a set of diverse products. In particular, the suitability of the proposed NS is evaluated on 31 FMs, being either real or artificial, with respect to the t-wise coverage. To investigate the scalability of the proposed method, we consider small-, moderate- and large-scale models and five values of t (i.e., $t = 2, \ldots, 6$), following the common practice in [4]. Moreover, the NS algorithm is comprehensively compared with three highly-related approaches [4], [7], and all of them are able to well scale to large FMs and high interaction strengths. Finally, we investigate how NS is affected by the way of generating new configurations and by its key parameters.

In summary, this paper provides the following contributions:

1) We perform a correlation analysis to explore the internal relationship between similarity metrics and

the t-wise coverage, observing that similarity metrics have a *significantly positive correlation* with the t-wise coverage. The above finding, which has not been revealed before, is important as it forms the theoretical foundation of the similarity-based testing (as a surrogation of the t-wise testing).

2) We demonstrate that NS is inherently an ideal tool for the similarity-based testing of SPLs. We show that, compared with the fitness function used in GA [4], the novelty score adopted by NS has *a (much) stronger positive correlation* with the t-wise coverage in most cases. This potentially implies the superiority of NS over GA in achieving higher t-wise coverage. According to our experiments, NS indeed performs better than GA, as well as other related algorithms.
3) In search-based approaches, the *unpredictable* way [4], implemented by the randomized SAT4J solver, forms a state-of-the-art strategy for generating new products in the SPL field [8], [57], [58], [59], [60], [61]. In this paper, we empirically demonstrate that combining two types of satisfiability solvers (i.e., the randomized SAT4J[1] + probSAT[2] [63]) in a similar way as in [59], [60] can be more effective than the *unpredictable* way when generating new products on some large-scale real-world FMs, such as the Linux kernel model (2.6.28.6-icse11) [29]. We consider this as one of the contributions because it provides a powerful alternative to generating new products in the domain of SPL testing.

The remainder of this paper is organized as follows. Section 2 presents concepts and notations used in this work. Section 3 details the proposed method, and Section 4 reports on the empirical study. In Section 5, we discuss threats to validity before reviewing related work in Section 6. Finally, Section 7 concludes the paper and outlines possible research lines for future studies.

2 Concepts and Notations

In this section we present background concepts and notations used in this paper.

2.1 Feature model configurations as test cases

This paper focuses on model-based testing of SPLs in which an FM [2] is used as the variability model. An FM is tree-like structure encompassing a set of features and constraints among them. Since feature modeling is a popular way of documenting commonality and variability of an SPL in both academic and industrial communities [4], [10], [64], it makes sense to develop FM-based SPL testing techniques.

Definition 1. *A **feature model** can be seen as a tuple $(\mathcal{F}, \mathcal{C})$ [4], where $\mathcal{F} = (f_1, \ldots, f_n)$ represents a set of n Boolean features (or configuration options), and $\mathcal{C} = (c_1, \ldots, c_k)$ is a set of k constraints among these features. $\mathcal{C}$ is satisfied if and only if all c_i $(i = 1, \ldots, k)$ are evaluated to True. Each c_i is called a clause if the feature model is transformed to a Boolean formula in conjunctive normal form (CNF) [65].*

In this context, we define a configuration of an FM as a set of selected/deselected features. Formally, we have the following definition.

Definition 2. *A **configuration** is a set $CF = \{\pm f_1, \ldots, \pm f_n\}$, where $+f_i$ and $-f_i$ indicate that the feature f_i is selected and deselected in the current configuration, respectively. Note that a feature can be either selected or deselected. Moreover, a configuration is said to be* valid *if and only if all the FM constraints are satisfied. Otherwise, it is called an* invalid *configuration.*

In the program, a configuration is encoded as a binary vector, and each of the elements takes either 1 (selected) or 0 (deselected). Notice that a configuration can be also called a *product* in the SPL terminology, an *individual* (or a *solution/point*) in the evolutionary computation (EC) terminology. In essence, these terms refer to the same thing, and thus they are used interchangeably in this paper.

Definition 3. *A **test suite** of an SPL is a list $TS = CF_1, \ldots, CF_N$, where each CF_i $(i = 1, \ldots, N)$ is a valid configuration.*

In this context, the valid configuration CF_i is called a **test case**.

2.2 T-wise testing and coverage

The t-wise testing for SPLs focuses on the combinations of $t \geq 2$ features of an SPL [19], [66]. It considers all the possible interactions between selected and deselected features. We list the following definitions related to the t-wise testing.

Definition 4. *A **t-set** is a set $\{\pm f_1, \ldots, \pm f_t\}$, where $t \leq n$. It represents a partially configured product [13].*

Definition 5. *A t-set is called a **valid t-set** if it satisfies the constraint $\mathcal{C}$ of the FM. A t-set that does not satisfy $\mathcal{C}$ is said to be invalid.*

A valid t-set, ts, is covered by a configuration CF, if $ts \subseteq CF$. Note that invalid t-sets need not to be covered [7].

Definition 6. *The **t-wise coverage** of a test suite $TS = CF_1, \ldots, CF_N$ is defined as the ratio $\frac{|\cup_{i=1}^{N} \mathcal{VT}_{CF_i}|}{|\mathcal{VT}_{fm}|}$, where $\mathcal{VT}_{fm}$ is the set of all the valid t-sets of the given FM; $\mathcal{VT}_{CF_i}$ denotes the set of t-sets covered by the configuration CF_i, and $|\cdot|$ returns the cardinality of a set. For simplicity, we also refer to* t-wise coverage *as* coverage.

3 The Novelty Search for SPL Testing

The NS algorithm for test case generation in SPL testing, as outlined in Algorithm 1, aims at looking for a specified amount of diverse configurations, using the specified amount of time. The key idea behind this algorithm is to use "novelty" for the selection of promising configurations. In the algorithm, the following control parameters need to be specified before the algorithm begins. The first two parameters are common in search-based algorithms, while the remaining two ones are unique to the NS algorithm.

1. The SAT4J [62] is a conflict-driven clause learning (CDCL) solver.
2. The probSAT is a stochastic local search (SLS) solver.

- The archive size (N) specifies the number of configurations to be returned. The setting of this parameter depends largely on the demands of software testers.
- The execution time allowed (max_t) serves as the termination condition of the algorithm. Similar to N, this parameter is preset by software testers.
- The repair probability (P_r) is introduced when generating new configurations. A random configuration is repaired, with the probability P_r, using the probSAT solver [63]. Proper values of this parameter will be experimentally investigated later in Section 4.4.1.
- The neighbor size (N_b) specifies the number of neighbors to be considered when evaluating the "novelty" of a configuration. Empirical studies will be conducted in Section 4.4.2 to tune this parameter.

Notice that the NS algorithm allows software testers flexibly specifying the number of configurations as well as the execution time. Such flexibility is desirable for the sampling algorithm to meet a given testing budget [4]. In the following subsections, we will give details on the main algorithmic components.

Algorithm 1 NS algorithm for SPL testing

Input: archive size (N), execution time allowed (max_t), repair probability (P_r) and neighbor size (N_b)
Output: archive $\mathcal{A}$

1: Initialize the archive $\mathcal{A}$ by generating N configurations in an *unpredictable* way as in [4]
2: **while** the elapsed time is less than max_t **do**
3: Generate a random configuration $\boldsymbol{c}$
4: **if** $\boldsymbol{c}$ is not valid **then**
5: **if** $rand(0,1) < P_r$ **then**
6: Repair $\boldsymbol{c}$ using the probSAT solver [63]
7: **else**
8: Replace $\boldsymbol{c}$ by an unpredictable configuration got from the randomized SAT4J solver
9: **end if**
10: **end if**
11: Calculate the novelty score $\rho(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathcal{A} \cup \{\boldsymbol{c}\}$
12: $\boldsymbol{c}_{worst} \leftarrow$ the worst member in $\mathcal{A}$ concerning the novelty score // $\boldsymbol{c}_{worst}$ *has the minimum novelty score*
13: **if** $\rho(\boldsymbol{c}) > \rho(\boldsymbol{c}_{worst})$ **then**
14: $\boldsymbol{c}_{worst} \leftarrow \boldsymbol{c}$ // *Replace the worst archived member by* $\boldsymbol{c}$
15: **end if**
16: **end while**
17: **return** $\mathcal{A}$

3.1 Initialization of the archive

According to Line 1 of Algorithm 1, $\mathcal{A}$ is initialized with N valid configurations produced in an *unpredictable* way [4]. Since an FM can be converted into a Boolean formula [65], an off-the-shelf satisfiability (SAT) solver, such as SAT4J[3] [62], can be naturally used to generate valid products. However, the internal order used by the solver to parse the logical clauses and literals typically produces the same solution in a deterministic way, resulting in the loss of diversity of configurations. To overcome this issue, Henard et al. [4] proposed to randomize the order how the logical clauses and the literals are parsed by the solver. In the implementation, they used the randomized SAT4J solver [62]. This way, it is impossible to predict the next product to be returned. In other words, products are generated in an unpredictable way. Experimental results have shown that this unpredictable way can significantly improve the diversity of the configurations [4], [58].

Being simple and effective, the above *unpredictable* way forms a state-of-the-art strategy to sample configurations from the space of all the valid products [57]. In fact, this strategy has been widely used to generate configurations in the context of both optimal software product selection [58], [59], [60] and software product line testing [8], [61].

To initialize the archive $\mathcal{A}$, we also adopt this *unpredictable* way to generate configurations. Notice that, to further improve diversity, the archive only allows the entry of configurations that are different from the already archived ones.

3.2 Generation of new configurations

In the iteration process (Lines 3-15 of Algorithm 1), the procedure tries to update $\mathcal{A}$ by generating a new valid configuration each time. To this end, we first produce a random configuration $\boldsymbol{c}$, with each feature being either selected or deselected both with a probability 0.5. Due to constraints among features, the randomly generated configuration is very likely to be invalid. In this case, it is either repaired by the probSAT solver [63] with the probability P_r (see Line 6), or replaced by an unpredictable configuration obtained from the randomized SAT4J solver [62] (see Line 8).

3.2.1 Repair configurations using probSAT

The procedure of probSAT [63], an SAT solver based on the stochastic local search (SLS)[4] [68], is given in Algorithm 2. Essentially, it iteratively flips a selected variable (by changing its value from 0 to 1, or vice verse) until the number of flips reaches the maximum value, max_flips. For an invalid configuration $\boldsymbol{c}$, it violates at least one clause. As shown in Line 3 of Algorithm 2, we randomly pick one of these falsified clauses, denoted by cls. Next, the procedure works out the *break* value of each variable v in cls (Line 5 in Algorithm 2). The *break* value of a variable is defined as the number of satisfied clauses that would become falsified after flipping this variable [69]. It is clear from this definition that it would be better (in a general case) to flip the variable with smaller *break* value. To efficiently compute this value, we adopt the fast procedure proposed in [69]. For details, please refer to the original study.

Subsequently, as shown in Line 6, the $break\ (v)$ is transformed by the following polynomial function[5].

3. https://www.sat4j.org/

4. In addition to SLS-style solvers, another mainstream of high-performance algorithms for satisfiability solving is conflict-driven clause learning (CDCL) solvers [67], such as SAT4J. Different from CDCL-style solvers which need to methodically traverse the whole search space, SLS-style solvers are typically greedy algorithms aiming at quickly satisfying clauses as many as possible. Therefore, SLS-style solvers are in general computationally efficient. However, they have no guarantee on finding a satisfying solution for each solver call.

5. According to [63], $break\ (v)$ can be transformed by other types of functions. The polynomial function is chosen in this study primarily because of its simplicity and effectiveness [63].

$$f(v) = [\epsilon + break(v)]^{-c_b}, \quad (1)$$

where $\epsilon = 1$, and $c_b > 1.0$. Clearly, $f(v)$ monotonically decreases with respect to $break(v)$. This means that we tend to choose the variables with large $f(v)$.

Algorithm 2 Procedure of the probSAT solver

Input: an invalid configuration ($\boldsymbol{c}$), maximum number of flips (max_flips)
Output: the repaired $\boldsymbol{c}$

1: $n \leftarrow 0$ // the number of flips
2: **while** $n \leq max\_flips$ **do**
3: Randomly pick a falsified clause cls with respect to the configuration $\boldsymbol{c}$
4: **for** each variable v in cls **do**
5: Compute the *break* value of v, i.e., $break\ (v)$
6: Transform $break\ (v)$ according to Eq. (1)
7: **end for**
8: $var \leftarrow$ the variable selected based on the probability $\frac{f(var)}{\sum_{z \in cls} f(z)}$
9: Flip the variable var in $\boldsymbol{c}$
10: $n++$
11: **end while**
12: **return** $\boldsymbol{c}$

Finally, according to Line 8 of Algorithm 2, the variable var to be flipped is chosen based on the probability $\frac{f(var)}{\sum_{z \in cls} f(z)}$. This calculation of probability ensures that variables with larger f values are given more chances to be selected. The chosen variable is then flipped, coming with the counter n increased by 1 (Lines 9 and 10).

It should be mentioned that repairing SPL configurations using probSAT has already been explored in our previous work [60] in the context of multi-objective software product selection from SPLs. It has been shown that probSAT is more effective than WalkSAT [70], another popular SLS-style solver, in search for dissimilar products. Since this fits well our goal in this study (generating SPL configurations as dissimilar as possible), we choose probSAT instead of WalkSAT. Moreover, the possibility of using SLS-style solvers to repair SPL configurations has not yet been explored in the context of SPL testing. We make the first step in this regard. Most importantly, as will be demonstrated in Section 3.2, the introduce of probSAT-based repair operator can significantly improve the coverage on some FMs, like for example the Linux kernel model (2.6.28.6-icse11) [29] which is the largest FM widely studied before.

In the detailed implementation, max_flips is set to 4,000, following the common practice in [59], [60]. The parameter c_b in (1) is set to 2.165, and this value is suggested by the developers of probSAT. Since we use standard values from the literature for the two parameters, a tuning phase is not required in this case.

3.2.2 Replacement based on randomized SAT4J solver

In case that the random configuration $\boldsymbol{c}$ is invalid, it will be replaced, with the probability $1 - P_r$, by an unpredictable configuration obtained from the randomized SAT4J solver [4]. As discussed previously in Section 3.1, the randomized solver enables an exploration of the valid search space in an *unpredictable* way, being capable of improving diversity of configurations. To implement the randomized solver, following the common practice in [4], we primarily randomize the order in which the assignments (*true* or *false*) to literals are instantiated. More specifically, literals are assigned with either *true* (i.e., 1) or *false* (i.e., 0) both with a probability 0.5. That way, it prevents from generating biased configurations towards either *true* or *false* assignment.

3.3 Evaluation of novelty

According to Line 11 in Algorithm 1, the generated configuration $\boldsymbol{c}$ (after repair or replacement) and all the archived members are jointly evaluated for novelty. For any $\boldsymbol{x} \in \mathcal{A} \cup \{\boldsymbol{c}\}$, the novelty score of $\boldsymbol{x}$ is given by

$$\rho(\boldsymbol{x}) = \frac{1}{N_b} \sum_{i=1}^{N_b} d(\boldsymbol{x}, \boldsymbol{\mu}_i), \quad (2)$$

where N_b is the neighbor size, and $\boldsymbol{\mu}_i$ is the i-th nearest neighbor of $\boldsymbol{x}$ with respect to the following Jaccard distance [71].

$$d(\boldsymbol{x}, \boldsymbol{\mu}_i) = 1 - \frac{|\boldsymbol{x} \cap \boldsymbol{\mu}_i|}{|\boldsymbol{x} \cup \boldsymbol{\mu}_i|}, \quad (3)$$

where $|\cdot|$ denotes the cardinality of a set. Since both $\boldsymbol{x}$ and $\boldsymbol{\mu}_i$ are sets according to Definition 2, $d(\boldsymbol{x}, \boldsymbol{\mu}_i)$ is a set-based distance. Clearly, the distance varies between 0 and 1. In particular, a distance of 0 indicates two identical configurations, while a distance equal to 1 suggests that the two considered configurations are totally different. In fact, this distance metric was adopted in [4] to measure the degree of similarity between two configurations. Given its good performance presented in the prior work [4], we choose it in our study as a measure of behavioral difference between two configurations in the search space.

According to (2), the novelty score is defined as the average distance to the N_b-nearest neighbors of a configuration, where N_b is a fixed parameter. In fact, the novelty score estimates the sparseness of a point in the decision space. If the score of a given point (or configuration) is large, then it is in a sparse area; in contrast, it is in a dense area in case that the novelty score is small.

It is important to note that the novelty score measures how unique an individual's behaviour is, with respect to the behaviours of both archived individuals and the current one that represents the most recently visited point [50]. This is reason why we consider the set $\mathcal{A} \cup \{\boldsymbol{c}\}$, instead of $\mathcal{A}$, when evaluating the novelty of an individual.

3.4 Reward novel individuals

As shown in Line 12 of Algorithm 1, the worst member in $\mathcal{A}$, $\boldsymbol{c}_{worst}$, is identified by comparing novelty scores. After that, $\boldsymbol{c}_{worst}$ will be replaced by the newly generated $\boldsymbol{c}$ in case that $\rho(\boldsymbol{c}) > \rho(\boldsymbol{c}_{worst})$. The above operation simply aims at rewarding "novel" individuals, creating a constant pressure to do something new. Indeed, the new individual that is far away from its predecessors takes place of the archived one with the least novelty. That way, the search is driven toward unexplored regions, enabling a diverse exploration of the search space.

It should be noted that no explicit objective is used in the NS algorithm. Instead, by attempting to maximize the novelty metric defined in the decision space, the algorithm tends to generate dissimilar configurations. This is precisely what we pursue in SPL testing.

3.5 Why Novelty Search?

As discussed in [53], NS is well suitable for SBSE problems. Going one step further, we demonstrate in this paper that NS is an ideal tool for the similarity-based testing of SPLs because of the following truths.

- In SPL testing, a set of test cases are required to cover valid t-sets as many as possible. The NS algorithm maintains an archive which stores the first N most novel individuals found during the whole search process. These individuals can directly serve as test cases for the SPLs.
- The results presented in [4] suggest that *two dissimilar configurations are more likely to cover a greater number of valid t-sets than two similar ones*. Therefore, the goal of similarity-based SPL testing is essentially to find a set of configurations as diverse (or dissimilar) as possible. In fact, different from fitness-oriented evolutionary algorithms which drive the individuals towards peaks of fitness, the NS is born to achieve this goal since it constantly searches for "novel" (a synonym for "diverse") individuals. This way, the diversity of the population can be naturally improved.
- Previous works [4], [9] on SPL testing ambiguously adopt the idea of NS, i.e., searching for diverse configurations using heuristics like GA to optimize similarity-based fitness function. In this paper, we explicitly use NS because of its good theoretical properties, e.g., *behaving like a uniform random search process in the behavior space* [51] and *creating a pressure for high evolvability even in bounded behavior spaces* [72]. These properties are very useful in guiding the search towards diverse configurations in the testing of SPLs.

Experiments presented in the next section will show that NS can indeed obtain promising results in testing SPLs with respect to the t-wise coverage.

4 Empirical Study

The empirical study conducted in this section aims at answering the following four research questions.

- **RQ1**: *What is the relationship between similarity metrics and the t-wise coverage*?
- **RQ2**: *How does the NS perform compared with state-of-the-art algorithms for similarity-based testing of SPLs*?
- **RQ3**: *Do ways of generating new configurations make any difference in similarity-based testing of SPLs*?
- **RQ4**: *How is the performance of NS affected by its key parameters*?

The first research question is a foundational question in similarity-based testing of SPLs. Indeed, if similarity metrics have no relationships with the t-wise coverage, then it is meaningless to optimize the similarity metrics so as to achieve a decent t-wise coverage. To address RQ1, a correlation analysis is performed to investigate the internal relationship between two similarity metrics (i.e., the similarity-based fitness function [4] and the novelty score) and the t-wise coverage. The second research question amounts to evaluating the NS algorithm in comparison with some state-of-the-art approaches. We expect the NS to provide a t-wise coverage better than or close to the state of the art. The third research question aims at comparing different ways of generating new configurations in similarity-based SPL testing. Currently, the *unpredictable* way proposed by Henard et al. [4] forms a state-of-the-art strategy for new individuals generation. However, it remains unknown whether this strategy can be further improved. Finally, the fourth research question seeks to provide useful guidelines for tuning parameters in the NS algorithm. Since this is the first work which adopts NS for the testing of SPLs, it is of practical importance to give some suggestions on the setting of its key parameters.

The conducted experiments are performed on a Quad Core@2.20 GHz with 8 GB of RAM. As shown in Table 1, this study employs 31 FMs which are divided into three categories. The first category is composed of 12 FMs with the number of features lower than 200; they are referred to as *small*-scale FMs. The second category consists of 13 models (with the number of features larger than 200 but lower than or equal to 1,000), and they are referred to as *moderate*-scale FMs. The third category contains 6 models of large size (beyond 1,000); they are referred to as *large*-scale FMs. For each FM, Table 1 presents its name, the number of features, the number of CNF constraints, the number of free features and the number of valid 2-sets.

A free feature is the feature whose assignment is undetermined, while a fixed feature can be either *mandatory* or *dead*. A *mandatory* feature must be presented in any valid configuration, whereas a *dead* feature cannot be included in any case. It is the number of free features that determines the size of the search space. According to Table 1, the number of free features ranges from the smallest 18 to the largest 16,664.

For small- and moderate-scale FMs and for $t = 2$, we enumerate all the possible t-sets and ask an SAT solver to determine whether they are valid or not. For the large FMs, it is very time-consuming to obtain all the valid t-sets even for $t = 2$ [4]. To compute the t-wise coverage in this case, we instead randomly generate a set of 10,000 valid t-sets, which can be seen as a sample of the whole space of all the valid t-sets. Since the number of valid t-sets explodes as t increases, we sample 10,000 valid t-sets for the FMs when $t \geq 3$. The number of valid 2-sets is listed in the last column in Table 1.

Finally, notice that most of the FMs have been chosen by Henard et al. [4] in their empirical study, and they are taken from either the SPLOT repository [73][6] or the LVAT repository [29][7]. In this work, we add three moderate FMs, i.e., E-shop from SPLOT; toybox and axTLS from LVAT, and

6. SPLOT: http://www.splot-research.org/
7. LVAT: http://code.google.com/p/linux-variability-analysis-tools

TABLE 1
Feature models used in the empirical study

	Feature model	#Features	#Constraints	#Free Features	#Valid 2-sets
Small	CounterStrikeSimpleFM	24	35	18	833
	SPLSSimuelESPnP	32	54	23	1,448
	DSSample	41	201	34	2,592
	WebPortal	43	68	39	3,196
	Drupal	48	79	40	3,751
	ElectronicDrum	52	119	36	3,746
	SmartHomev2.2	60	82	53	6,189
	VideoPlayer	71	99	53	7,528
	Amazon	79	250	73	10,555
	ModelTransformation	88	151	76	13,139
	CocheEcologico	94	191	57	11,075
	Printers	172	310	122	42,638
Moderate	E-shop	290	426	260	149,723
	toybox	544	1,020	175	256,494
	axTLS	684	2,155	300	476,386
	SPLOT-Generated-FM-1000-1	1,000	1,875	949	1,861,476
	SPLOT-Generated-FM-1000-2	1,000	1,927	979	1,939,079
	SPLOT-Generated-FM-1000-3	1,000	1,933	966	1,913,540
	SPLOT-Generated-FM-1000-4	1,000	1,807	970	1,929,827
	SPLOT-Generated-FM-1000-5	1,000	1,889	994	1,973,526
	SPLOT-Generated-FM-1000-6	1,000	1,814	990	1,968,013
	SPLOT-Generated-FM-1000-7	1,000	1,874	967	1,919,404
	SPLOT-Generated-FM-1000-8	1,000	1,897	880	1,742,123
	SPLOT-Generated-FM-1000-9	1,000	1,788	846	1,688,239
	SPLOT-Generated-FM-1000-10	1,000	1,935	982	1,952,788
Large	ecos-icse11	1,244	3,146	1,189	*10,000*
	freebsd-icse11	1,396	62,183	1,355	*10,000*
	Automotive01	2,513	10,311	2,218	*10,000*
	SPLOT-Generated-FM-5000	5,000	9,419	4,925	*10,000*
	2.6.28.6-icse11	6,888	343,944	6,728	*10,000*
	Automotive02 V3	18,434	347,557	16,664	*10,000*

two large FMs, i.e., Automotive01[8] and Automotive02 [25], [31][9]. The two large FMs are closed-source product lines from automotive industry, and are well suited as subjects for examining the scalability of t-wise testing algorithms [25]. Regarding the FMs, 20 of them are real, while 11 are artificially generated by the SPLOT FM generator [73], with the prefix being "SPLOT-Generated-FM" in their names.

4.1 Correlation analysis between similarity metrics and t-wise coverage (RQ1)

In this section, we perform a correlation analysis to investigate the relationship between two similarity metrics and the t-wise coverage. We first briefly introduce the two similarity metrics, and then present results of the correlation analysis. Finally, answers to RQ1 are given.

4.1.1 Similarity metrics under study

Two similarity metrics are considered. Given a test suite with N configurations, i.e., $TS = CF_1, \ldots, CF_N$, both similarity metrics map a test suite to a positive real value. Formally, the first one, as given in Eq. (4), is called the similarity-based fitness function [4].

$$f(TS) = \sum_{j>i}^{N} d(CF_i, CF_j), \tag{4}$$

where $d(CF_i, CF_j)$ calculates the Jaccard distance between CF_i and CF_j (see Eq. (3)). This similarity metric sums the Jaccard distance over all pairs of configurations such that $j > i$.

The second one is formulated as follows:

$$\rho(TS) = \sum_{i=1}^{N} \rho(CF_i), \tag{5}$$

where $\rho(CF_i)$ is the novelty score of a single configuration (see Eq. (2)). This function generalizes the calculation of novelty score from a single configuration to a test suite TS.

Intuitively, according to [4], the higher both similarity metrics of a given TS, the higher the distance among configurations, leading to potentially larger t-wise coverage. However, this has not been rigorously verified. Since it may be very difficult to directly prove the above argument, we instead statistically demonstrate it by performing a correlation analysis. To this end, we generate 100 samples for each FM, with a sample being a test suite of 100 configurations. Note that these configurations are generated using the *unpredictable* strategy suggested by Henard et al. [4]. For each sample, we can calculate its t-wise coverage, as well as the values of the two similarity metrics. This way, we obtain 100 pairs of data for each FM regarding each similarity metric, enabling us to perform a correlation analysis to observe how and, to what extend, the t-wise coverage is correlated with the similarity metrics.

8. Automotive01 is integrated in the FeatureIDE [74]: https://featureide.github.io/

9. https://github.com/PettTo/SPLC2019_The-Scalability-Challenge_Product-Lines/tree/master/Automotive02

TABLE 2
Results of the correlation analysis performed on small-scale FMs. For r, larger values are shown in **bold**; For p, values larger than $\alpha = 0.05$ are underlined.

Feature model	t-wise	r		p	
		Similarity-based fitness	Novelty score	Similarity-based fitness	Novelty score
CounterStrikeSimpleFM	2	**0.0594**	0.0109	0.5571	0.9141
	3	0.1798	**0.1928**	0.0734	0.0546
	4	**0.3986**	0.3583	0.0000	0.0003
	5	0.4490	**0.5556**	0.0000	0.0000
	6	0.4680	**0.6732**	0.0000	0.0000
SPLSSimuelESPnP	2	**0.0899**	0.0479	0.3739	0.6358
	3	0.5782	**0.6970**	0.0000	0.0000
	4	0.6711	**0.8395**	0.0000	0.0000
	5	0.6973	**0.9040**	0.0000	0.0000
	6	0.6715	**0.9449**	0.0000	0.0000
DSSample	2	0.1716	**0.2277**	0.0878	0.0227
	3	0.3012	**0.3517**	0.0023	0.0003
	4	0.2645	**0.3787**	0.0078	0.0001
	5	0.3388	**0.4928**	0.0006	0.0000
	6	0.4001	**0.5322**	0.0000	0.0000
WebPortal	2	**0.4806**	-0.0475	0.0000	0.6389
	3	**0.6900**	0.3815	0.0000	0.0001
	4	**0.7064**	0.4494	0.0000	0.0000
	5	**0.7281**	0.6498	0.0000	0.0000
	6	0.6254	**0.7144**	0.0000	0.0000
Drupal	2	**0.1216**	0.1008	0.2280	0.3183
	3	**0.4768**	0.3098	0.0000	0.0017
	4	**0.6855**	0.5851	0.0000	0.0000
	5	**0.8087**	0.7066	0.0000	0.0000
	6	**0.7425**	0.7318	0.0000	0.0000
ElectronicDrum	2	**0.2416**	0.1549	0.0154	0.1238
	3	0.4964	**0.5268**	0.0000	0.0000
	4	**0.6892**	0.6622	0.0000	0.0000
	5	0.7091	**0.7687**	0.0000	0.0000
	6	**0.7366**	0.7239	0.0000	0.0000
SmartHomev2.2	2	**0.1783**	-0.0473	0.0759	0.6400
	3	**0.5075**	0.1731	0.0000	0.0850
	4	**0.6987**	0.4409	0.0000	0.0000
	5	**0.8048**	0.6245	0.0000	0.0000
	6	**0.7531**	0.6826	0.0000	0.0000
VideoPlayer	2	0.0589	**0.1231**	0.5603	0.2222
	3	0.4626	**0.5259**	0.0000	0.0000
	4	0.8300	**0.8603**	0.0000	0.0000
	5	0.9330	**0.9481**	0.0000	0.0000
	6	0.9536	**0.9656**	0.0000	0.0000
Amazon	2	-0.0313	**-0.0042**	0.7575	0.9666
	3	0.1015	**0.1434**	0.3151	0.1547
	4	0.0499	**0.1981**	0.6220	0.0482
	5	0.2678	**0.3733**	0.0071	0.0001
	6	0.2105	**0.3774**	0.0355	0.0001
ModelTransformation	2	0.6405	**0.7297**	0.0000	0.0000
	3	0.7925	**0.8853**	0.0000	0.0000
	4	0.8354	**0.9339**	0.0000	0.0000
	5	0.8493	**0.9531**	0.0000	0.0000
	6	0.8417	**0.9663**	0.0000	0.0000
CocheEcologico	2	**0.2460**	0.1986	0.0136	0.0476
	3	**0.4665**	0.4482	0.0000	0.0000
	4	0.4348	**0.7277**	0.0000	0.0000
	5	0.4921	**0.8163**	0.0000	0.0000
	6	0.5052	**0.9111**	0.0000	0.0000
Printers	2	0.3021	**0.3357**	0.0023	0.0006
	3	0.5668	**0.6554**	0.0000	0.0000
	4	0.6743	**0.8375**	0.0000	0.0000
	5	0.6931	**0.8995**	0.0000	0.0000
	6	0.6964	**0.9335**	0.0000	0.0000

4.1.2 Results of the correlation analysis

The correlation analysis returns the correlation coefficient r and the p-value for testing the hypothesis that there is no relationship between the two random variables (null hypothesis). The r measures the linear dependence between two variables, and its value can range from -1 to 1, with -1 representing a perfect negative correlation, 0 representing no correlation, and 1 representing a perfect positive correlation. The p-values range from 0 to 1, where values smaller than the significance level α (default is 0.05) indicate that the

TABLE 3
Results of the correlation analysis performed on moderate-scale FMs. For r, larger values are shown in **bold**; For p, values larger than $\alpha = 0.05$ are underlined.

Feature model	t-wise	r		p	
		Similarity-based fitness	Novelty score	Similarity-based fitness	Novelty score
E-shop	2	0.3125	**0.3669**	0.0016	0.0002
	3	0.6836	**0.7222**	0.0000	0.0000
	4	0.8769	**0.8975**	0.0000	0.0000
	5	0.9400	**0.9569**	0.0000	0.0000
	6	0.9639	**0.9694**	0.0000	0.0000
toybox	2	0.0249	**0.3083**	0.8059	0.0018
	3	0.1358	**0.6312**	0.1780	0.0000
	4	0.1363	**0.7360**	0.1763	0.0000
	5	0.1494	**0.7566**	0.1379	0.0000
	6	0.1840	**0.8710**	0.0669	0.0000
axTLS	2	0.5016	**0.5786**	0.0000	0.0000
	3	0.5679	**0.6677**	0.0000	0.0000
	4	0.5852	**0.7844**	0.0000	0.0000
	5	0.6017	**0.8128**	0.0000	0.0000
	6	0.6000	**0.8185**	0.0000	0.0000
SPLOT-Generated-FM-1000-1	2	0.4757	**0.5817**	0.0000	0.0000
	3	0.5443	**0.8106**	0.0000	0.0000
	4	0.4839	**0.8932**	0.0000	0.0000
	5	0.5122	**0.9435**	0.0000	0.0000
	6	0.5187	**0.9235**	0.0000	0.0000
SPLOT-Generated-FM-1000-2	2	0.5943	**0.7623**	0.0000	0.0000
	3	0.6383	**0.8706**	0.0000	0.0000
	4	0.6429	**0.9296**	0.0000	0.0000
	5	0.6275	**0.9407**	0.0000	0.0000
	6	0.6247	**0.9555**	0.0000	0.0000
SPLOT-Generated-FM-1000-3	2	0.5285	**0.7239**	0.0000	0.0000
	3	0.5659	**0.8560**	0.0000	0.0000
	4	0.5287	**0.9134**	0.0000	0.0000
	5	0.5330	**0.9336**	0.0000	0.0000
	6	0.5301	**0.9457**	0.0000	0.0000
SPLOT-Generated-FM-1000-4	2	0.7002	**0.8651**	0.0000	0.0000
	3	0.7327	**0.9450**	0.0000	0.0000
	4	0.7565	**0.9692**	0.0000	0.0000
	5	0.7461	**0.9723**	0.0000	0.0000
	6	0.7242	**0.9700**	0.0000	0.0000
SPLOT-Generated-FM-1000-5	2	0.6545	**0.7784**	0.0000	0.0000
	3	0.6824	**0.8900**	0.0000	0.0000
	4	0.7255	**0.9292**	0.0000	0.0000
	5	0.6946	**0.9599**	0.0000	0.0000
	6	0.6939	**0.9524**	0.0000	0.0000
SPLOT-Generated-FM-1000-6	2	0.7167	**0.8317**	0.0000	0.0000
	3	0.7397	**0.9012**	0.0000	0.0000
	4	0.7552	**0.9400**	0.0000	0.0000
	5	0.7229	**0.9453**	0.0000	0.0000
	6	0.7357	**0.9553**	0.0000	0.0000
SPLOT-Generated-FM-1000-7	2	0.6122	**0.7781**	0.0000	0.0000
	3	0.6917	**0.9011**	0.0000	0.0000
	4	0.6857	**0.9329**	0.0000	0.0000
	5	0.6947	**0.9627**	0.0000	0.0000
	6	0.6729	**0.9584**	0.0000	0.0000
SPLOT-Generated-FM-1000-8	2	0.4667	**0.6720**	0.0000	0.0000
	3	0.5268	**0.8243**	0.0000	0.0000
	4	0.5081	**0.8879**	0.0000	0.0000
	5	0.4959	**0.9322**	0.0000	0.0000
	6	0.4683	**0.9390**	0.0000	0.0000
SPLOT-Generated-FM-1000-9	2	0.6020	**0.7874**	0.0000	0.0000
	3	0.6394	**0.9220**	0.0000	0.0000
	4	0.6639	**0.9373**	0.0000	0.0000
	5	0.6718	**0.9600**	0.0000	0.0000
	6	0.6611	**0.9694**	0.0000	0.0000
SPLOT-Generated-FM-1000-10	2	0.3864	**0.7449**	0.0000	0.0000
	3	0.4542	**0.8776**	0.0000	0.0000
	4	0.5377	**0.9450**	0.0000	0.0000
	5	0.5307	**0.9532**	0.0000	0.0000
	6	0.5143	**0.9669**	0.0000	0.0000

TABLE 4
Results of the correlation analysis performed on large-scale FMs. For r, larger values are shown in **bold**; For p, values larger than $\alpha = 0.05$ are underlined.

Feature model	t-wise	r		p	
		Similarity-based fitness	Novelty score	Similarity-based fitness	Novelty score
ecos-icse11	2	**0.7063**	0.5451	0.0000	0.0000
	3	**0.8388**	0.7853	0.0000	0.0000
	4	0.8969	**0.8984**	0.0000	0.0000
	5	0.8951	**0.9490**	0.0000	0.0000
	6	0.8766	**0.9647**	0.0000	0.0000
freebsd-icse11	2	0.5171	**0.5201**	0.0000	0.0000
	3	0.8260	**0.8266**	0.0000	0.0000
	4	0.9161	**0.9484**	0.0000	0.0000
	5	0.9389	**0.9665**	0.0000	0.0000
	6	0.9313	**0.9585**	0.0000	0.0000
Automotive01	2	0.6649	**0.8067**	0.0000	0.0000
	3	0.7074	**0.8961**	0.0000	0.0000
	4	0.6947	**0.9061**	0.0000	0.0000
	5	0.6887	**0.9110**	0.0000	0.0000
	6	0.6942	**0.9292**	0.0000	0.0000
SPLOT-Generated-FM-5000	2	0.5219	**0.7741**	0.0000	0.0000
	3	0.5681	**0.8757**	0.0000	0.0000
	4	0.5358	**0.9348**	0.0000	0.0000
	5	0.5364	**0.9468**	0.0000	0.0000
	6	0.5567	**0.9465**	0.0000	0.0000
2.6.28.6-icse11	2	0.6590	**0.7707**	0.0000	0.0000
	3	0.7537	**0.9521**	0.0000	0.0000
	4	0.7704	**0.9794**	0.0000	0.0000
	5	0.7537	**0.9795**	0.0000	0.0000
	6	0.7532	**0.9822**	0.0000	0.0000
Automotive02_V3	2	0.2012	**0.2150**	0.0447	0.0317
	3	0.2895	**0.3999**	0.0035	0.0000
	4	**0.1981**	0.1965	0.0482	0.0501
	5	0.3633	**0.5469**	0.0002	0.0000
	6	0.3981	**0.6013**	0.0000	0.0000

corresponding correlation is considered significant. Table 2 summarizes results of the correlation analysis performed on small-scale FMs. In almost all the cases, as seen, both similarity metrics show a positive correlation with the t-wise coverage. Moreover, most of the correlations are statistically significant since $p < 0.05$. In particular, it is observed that values of p are equal or very close to 0 in most cases. Going one step further, we work out the percentage of cases in which significant correlations are observed. It is 83.3 % for the similarity-based fitness, and 81.7% for the novelty score. From this perspective of view, the two metrics have no obvious differences. However, the novelty score shows better (or larger) r values than the similarity-based fitness in 63.3 percent of all the cases, and worse in only 36.7 percent. This implies that the novelty score would be more suitable than the similarity-based fitness as a substitution of the t-wise coverage.

In a similar way, we perform a correlation analysis on all the moderate-scale FMs, and tabulate the results in Table 3. Regarding the novelty score, it is observed that all p-values are smaller than 0.05. Regarding the similarity-based fitness, the corresponding correlations are considered significant on all the moderate-scale FMs except toybox. However, it can be found from the r values that the novelty score has a *(much) stronger positive correlation* with the t-wise coverage than its counterpart on all the feature models regardless of the value of t. Notice that r values for the novelty score are larger than 0.9 in a number of cases, which suggests that the novelty score and the t-wise coverage are almost linearly dependent.

The correlation analysis results on large-scale FMs are presented in Table 4, where we can find that both similarity metrics, just as in the moderate-scale case, have significantly positive correlations with the t-wise coverage on all large-scale FMs for all values of t, expect Automotive02 for $t = 4$. In this case, the p-value for the novelty score is sightly larger than 0.05. Moreover, as clearly reflected by the r values, the novelty score has a (much) stronger correlation with the t-wise coverage than the similarity-based fitness.

4.1.3 RQ1 summary

The correlation analysis performed in this section brings out the following conclusions. First, *both the similarity-based fitness and the novelty score have, in general, a significantly positive correlation with the t-wise coverage*. Moreover, this relationship is not affected by the size of feature models and by the t-wise strengths. The above findings explain why similarity can be used as a surrogate metric for the t-wise coverage. More importantly, it makes the similarity-based SPL testing theoretically solid. Second, *the novelty score has, in general, a (much) stronger positive correlation with the t-wise coverage than the similarity-based fitness suggested by Henard et al. [4].* It means that the novelty score should be more effective than the similarity-based fitness in guiding the search towards a set of configurations with a decent t-wise coverage. In the forthcoming subsection, we will empirically verify this by comparing NS with three highly-related approaches, one of which is Henard's GA algorithm developed on the basis of the similarity-based fitness function.

4.2 Comparison with state-of-the-art algorithms (RQ2)

This section focuses on demonstrating the effectiveness of NS through a comparative study. To begin with, we give a brief introduction to the algorithms under examination, and then we specify the experiment setup used in the empirical study. In what follows, experiment results are presented and conclusions are summarized.

4.2.1 Algorithms under comparison

We compare NS with three state-of-the-art algorithms, i.e., *Unpredictable* [4], *GA* [4] and *SamplingDown* [7]. Notice that we choose the above algorithms because: 1) they are all dedicated for similarity-based testing of SPLs, and 2) they are all able to scale to large FMs and high t-wise strengths.

- The *Unpredictable* approach, suggested by Henard et al. [4], uses the randomized SAT4J solver (see Section 3.2.2) to generate N configurations in an unpredictable way. This approach was chosen in [4] as a comparison basis because of its good scalability and high efficiency.
- The GA algorithm, also proposed by Henard et al. [4], may be the first one using a similarity-based heuristic to generate and prioritize product configurations for especially large SPLs. The algorithms starts with an initial population consisting of N configurations generated in the unpredictable way. Then they are prioritized using the global maximum distance prioritization method, and evaluated using the similarity-based fitness function defined by Eq. (4). The next step consists of attempting to replace the worst configuration by an unpredictable one got from the randomized SAT solver. This replacement is permitted if and only if the similarity-based fitness increases. The above step is repeated until the termination condition is satisfied. According to Henard et al. [4], this technique can be seen as a (1+1) genetic algorithm without crossover.
- The *SamplingDown* is adapted from the similarity-based prioritization proposed by Al-Hajjaji et al. [7]. The original approach, applied on any product sample, focuses solely on ordering products to increase the probability of detecting faults faster. In this paper, we extend it to an approach which samples down from a large set of unpredictable configurations. More specifically, the algorithm begins with a large initial set of configurations, then it selects candidates one by one based on similarity. First, the product with maximum number of features (or *all-yes-config* [75]) is selected. Then, we choose one by one the next product that has the minimum similarity with the already selected products. The process continues until N products are selected (N is often specified by software testers according to their test budgets).

4.2.2 Experiment setup

Experiment setups used in this section are summarized as follows.

- The number of returned configurations is 100 for all the algorithms, following the practice in [4].
- Each algorithm is independently run 30 times, and we present and analyze the experiment results regarding mean values and standard deviations of the obtained t-wise coverage.
- For both NS and GA, the termination of the algorithms can be flexibly controlled by specifying the maximum running time allowed (i.e., max_t). Both algorithms are allowed to run 6 seconds for small FMs, 30 seconds for moderate FMs, and 600 seconds for large FMs. These settings follow the practice in [59]. For Unpredictable and SamplingDown, their termination can not be manually controlled. Instead, they are automatically terminated once 100 configurations are generated/selected.
- In the NS algorithm, control parameters are set as follows: $P_r = 0.1$ and $N_b = 20$. The turning of the two parameters can be found later in Section 4.4. For the *SamplingDown* algorithm, the number of initial configurations is set to 1,000. For the remaining two algorithms, no control parameters are involved.

4.2.3 Experiment results

Table 5 presents mean values and standard deviations of the t-wise coverage obtained on the small-scale FMs. To determine whether the difference between NS and each of the peer algorithms is significant or not, we perform a Mann-Whitney U Test with a 0.05 significance level, following the guidelines suggested by Arcuri and Briand [76]. In particular, the symbols •, ‡ and ◦ indicate that NS performs better than, equivalently to and worse than the corresponding algorithms, respectively. Furthermore, the best mean results for each FM and each value of t are indicated by a dark-gray background, and the second-best results a light-gray background.

According to the summary at the bottom of Table 5, NS is the best-performing algorithm on small FMs, obtaining the best/second-best results in about $59/60 \approx 98\%$ cases. It is followed by SamplingDown which performs either best or second best in $35/60 \approx 58\%$ cases. Being competitive with SamplingDown, GA is able to obtain promising results in $31/60 \approx 52\%$ cases. Finally, the Unpredictable, the most ineffective algorithm, performs best in only two cases, and second best in only four cases. For moderate FMs, as shown in Table 6, NS is still the best algorithm, dramatically outperforming the other three algorithms. In particular, NS obtains the best average t-wise coverage in almost all the 65 cases except E-shop for $t = 2$ and $t = 3$, and SPLOT-Generated-FM-1000-3 for $t = 2$. It is observed that GA ranks second concerning the overall performance on moderate-scale FMs. As for the large-scale FMs, it can be found from Table 7 that NS and GA are highly competitive with each other, obtaining the best/second-best results in $29/30 \approx 97\%$ and $28/30 \approx 93\%$ cases, respectively. Clearly, both algorithms are overwhelmingly better than SamplingDown and Unpredictable, which are capable of obtaining the second-best results in only one and two cases, respectively.

Going one step further, we compute the percentage of cases in which NS performs better than (•), equivalently to (‡) and worse than (◦) each peer algorithm regarding all the FMs and all the t values considered (resulting in 155 cases in total). As shown in Table 8, NS shows a significant

TABLE 5
Mean values and standard deviations (in brackets) of the t-wise coverage on small-scale FMs for $t = 2, \ldots, 6$. The best and the second-best results are indicated by a dark-gray and a light-gray background, respectively.

Feature model	t-wise	NS	GA	SamplingDown	Unpredictable
CounterStrikeSimpleFM	2	100.00% (0.00)	100.00% (0.00)‡	100.00% (0.00)‡	99.988% (0.06)‡
	3	99.996% (0.01)	99.945% (0.03)•	99.989% (0.01)•	99.899% (0.20)•
	4	99.836% (0.05)	99.405% (0.11)•	99.667% (0.11)•	99.195% (0.37)•
	5	98.716% (0.14)	96.625% (0.31)•	98.166% (0.24)•	96.580% (0.67)•
	6	94.651% (0.26)	89.435% (0.51)•	93.771% (0.35)•	90.413% (0.86)•
SPLSSimuelESPnP	2	100.00% (0.00)	100.00% (0.00)‡	100.00% (0.00)‡	100.000% (0.00)‡
	3	100.00% (0.00)	99.993% (0.01)•	100.00% (0.00)‡	99.995% (0.01)•
	4	99.994% (0.01)	99.853% (0.06)•	99.966% (0.02)•	99.770% (0.08)•
	5	99.461% (0.09)	98.528% (0.16)•	99.229% (0.10)•	97.896% (0.22)•
	6	96.065% (0.21)	93.716% (0.29)•	95.610% (0.22)•	92.492% (0.34)•
DSSample	2	99.034% (0.12)	98.392% (0.14)•	98.188% (0.26)•	97.515% (0.50)•
	3	95.228% (0.27)	93.931% (0.33)•	93.615% (0.48)•	92.242% (0.72)•
	4	88.872% (0.41)	87.235% (0.42)•	87.037% (0.64)•	84.966% (0.94)•
	5	81.507% (0.47)	80.074% (0.57)•	79.894% (0.58)•	77.343% (0.68)•
	6	72.376% (0.52)	70.836% (0.48)•	70.820% (0.62)•	68.203% (0.88)•
WebPortal	2	99.998% (0.01)	99.999% (0.01)‡	99.969% (0.03)•	99.944% (0.06)•
	3	99.867% (0.04)	99.889% (0.03)○	99.636% (0.08)•	99.348% (0.21)•
	4	98.810% (0.10)	98.382% (0.17)•	97.892% (0.21)•	96.658% (0.51)•
	5	94.671% (0.26)	92.929% (0.40)•	92.861% (0.43)•	89.980% (0.76)•
	6	85.064% (0.30)	81.430% (0.52)•	82.972% (0.43)•	78.816% (0.86)•
Drupal	2	100.00% (0.00)	100.00% (0.00)‡	100.00% (0.00)‡	99.989% (0.02)•
	3	99.981% (0.02)	99.979% (0.02)‡	99.872% (0.05)•	99.771% (0.09)•
	4	99.622% (0.07)	99.528% (0.07)•	99.000% (0.13)•	98.302% (0.32)•
	5	96.936% (0.20)	96.142% (0.19)•	95.252% (0.25)•	93.160% (0.62)•
	6	88.719% (0.32)	86.821% (0.34)•	86.136% (0.42)•	82.976% (0.65)•
ElectronicDrum	2	100.00% (0.00)	100.00% (0.00)‡	100.00% (0.00)‡	99.992% (0.02)•
	3	99.856% (0.04)	99.656% (0.05)•	99.758% (0.06)•	99.447% (0.10)•
	4	98.489% (0.09)	97.740% (0.16)•	98.154% (0.14)•	97.038% (0.26)•
	5	94.495% (0.21)	92.926% (0.22)•	93.690% (0.26)•	91.744% (0.43)•
	6	87.887% (0.23)	85.706% (0.36)•	86.879% (0.28)•	83.956% (0.51)•
SmartHomev2.2	2	100.00% (0.00)	99.999% (0.00)‡	99.991% (0.01)•	99.964% (0.04)•
	3	99.952% (0.03)	99.951% (0.04)‡	99.822% (0.08)•	99.639% (0.13)•
	4	99.357% (0.08)	99.169% (0.12)•	98.620% (0.17)•	97.671% (0.27)•
	5	95.752% (0.21)	94.562% (0.28)•	93.825% (0.25)•	91.560% (0.39)•
	6	86.012% (0.33)	83.690% (0.38)•	83.346% (0.31)•	80.003% (0.55)•
VideoPlayer	2	100.00% (0.00)	100.00% (0.00)‡	100.00% (0.00)‡	100.000% (0.00)‡
	3	99.999% (0.00)	100.00% (0.00)‡	99.977% (0.02)•	99.985% (0.02)•
	4	99.834% (0.04)	99.909% (0.03)○	99.575% (0.09)•	99.693% (0.12)•
	5	98.346% (0.11)	98.180% (0.14)•	97.542% (0.28)•	97.366% (0.26)•
	6	92.232% (0.25)	91.302% (0.29)•	90.895% (0.33)•	90.012% (0.45)•
Amazon	2	94.548% (0.28)	93.922% (0.28)•	93.563% (0.61)•	92.136% (0.87)•
	3	86.478% (0.35)	85.240% (0.48)•	85.111% (0.61)•	83.022% (0.90)•
	4	77.475% (0.34)	75.823% (0.42)•	76.099% (0.65)•	73.598% (0.83)•
	5	68.142% (0.42)	66.090% (0.41)•	66.669% (0.59)•	64.004% (0.78)•
	6	60.215% (0.34)	58.146% (0.44)•	58.893% (0.61)•	56.175% (0.63)•
ModelTransformation	2	100.00% (0.00)	99.999% (0.00)‡	100.00% (0.00)‡	99.992% (0.02)•
	3	99.889% (0.03)	99.778% (0.05)•	99.798% (0.05)•	99.581% (0.14)•
	4	98.097% (0.13)	97.638% (0.16)•	97.590% (0.16)•	96.739% (0.31)•
	5	91.870% (0.25)	90.763% (0.24)•	90.773% (0.34)•	89.043% (0.47)•
	6	80.171% (0.30)	78.444% (0.42)•	78.878% (0.40)•	76.805% (0.54)•
CocheEcologico	2	99.772% (0.03)	99.682% (0.05)•	99.729% (0.04)•	99.566% (0.08)•
	3	98.672% (0.12)	98.288% (0.16)•	98.508% (0.10)•	97.908% (0.18)•
	4	96.079% (0.18)	95.220% (0.26)•	95.702% (0.17)•	94.499% (0.32)•
	5	91.168% (0.20)	89.953% (0.30)•	90.607% (0.26)•	88.848% (0.43)•
	6	85.345% (0.26)	83.583% (0.35)•	84.636% (0.35)•	82.328% (0.44)•
Printers	2	99.127% (0.28)	99.101% (0.29)‡	99.415% (0.14)○	99.214% (0.12)‡
	3	96.745% (0.35)	96.448% (0.36)•	96.994% (0.22)○	96.497% (0.27)•
	4	92.040% (0.31)	91.693% (0.39)•	92.189% (0.35)‡	91.462% (0.30)•
	5	84.843% (0.31)	84.228% (0.35)•	84.645% (0.31)•	83.754% (0.33)•
	6	75.299% (0.33)	74.697% (0.34)•	75.109% (0.27)•	74.177% (0.51)•
# Best or second best		59/60	31/60	35/60	6/60

improvement over GA, SamplingDown and Unpredictable in about 77%, 89% and 94% cases, respectively. Conversely, NS performs significantly worse in about 5%, 1% and 0% cases compared against the above three algorithms, respectively. These results imply that the performance of the algorithms follows the order: NS $>$ GA $>$ SamplingDown $>$ Unpredictable.

Since both NS and GA are search-based approaches, one may be interested in how the performance (regarding the t-wise coverage) changes during the search process. To this end, we record and plot curves of the t-wise coverage with respect to the running time on some representative

TABLE 6
Mean values and standard deviations (in brackets) of the t-wise coverage on moderate-scale FMs for $t = 2, \ldots, 6$. The best and the second-best results are indicated by a dark-gray and a light-gray background, respectively.

Feature model	t-wise	NS	GA	SamplingDown	Unpredictable
E-shop	2	99.986% (0.01)	99.990% (0.01)○	99.910% (0.04)●	99.917% (0.05)●
	3	99.697% (0.06)	99.741% (0.06)○	99.211% (0.15)●	99.159% (0.21)●
	4	97.679% (0.20)	97.570% (0.20)‡	96.282% (0.30)●	95.813% (0.45)●
	5	91.276% (0.23)	90.464% (0.31)●	88.861% (0.38)●	87.578% (0.69)●
	6	78.642% (0.41)	77.101% (0.41)●	75.820% (0.41)●	74.016% (0.64)●
toybox	2	100.00% (0.00)	100.00% (0.00)‡	100.00% (0.00)‡	100.00% (0.00)‡
	3	99.999% (0.00)	99.992% (0.01)●	99.980% (0.02)●	99.989% (0.01)●
	4	99.970% (0.02)	99.947% (0.02)●	99.889% (0.05)●	99.927% (0.03)●
	5	99.782% (0.05)	99.699% (0.06)●	99.496% (0.10)●	99.589% (0.08)●
	6	99.335% (0.08)	99.153% (0.08)●	98.867% (0.14)●	98.891% (0.12)●
axTLS	2	100.00% (0.00)	99.997% (0.00)●	99.989% (0.01)●	99.983% (0.01)●
	3	99.964% (0.02)	99.895% (0.03)●	99.807% (0.05)●	99.776% (0.07)●
	4	99.697% (0.05)	99.437% (0.08)●	99.146% (0.11)●	99.094% (0.13)●
	5	98.900% (0.11)	98.245% (0.11)●	97.698% (0.19)●	97.618% (0.25)●
	6	97.308% (0.16)	96.249% (0.23)●	95.426% (0.32)●	95.273% (0.28)●
SPLOT-Generated-FM-1000-1	2	97.460% (0.13)	97.022% (0.22)●	97.048% (0.16)●	96.755% (0.24)●
	3	91.871% (0.19)	90.795% (0.33)●	90.847% (0.29)●	90.097% (0.31)●
	4	81.864% (0.34)	80.298% (0.36)●	80.267% (0.24)●	79.151% (0.41)●
	5	68.815% (0.38)	66.926% (0.40)●	66.925% (0.34)●	65.633% (0.39)●
	6	54.639% (0.41)	52.573% (0.40)●	52.765% (0.41)●	51.458% (0.47)●
SPLOT-Generated-FM-1000-2	2	98.406% (0.07)	98.186% (0.14)●	98.163% (0.07)●	97.912% (0.20)●
	3	93.160% (0.22)	92.317% (0.28)●	92.505% (0.25)●	91.688% (0.33)●
	4	82.407% (0.35)	80.999% (0.32)●	81.218% (0.25)●	80.013% (0.37)●
	5	67.909% (0.35)	66.365% (0.36)●	66.573% (0.32)●	65.150% (0.45)●
	6	51.966% (0.37)	50.434% (0.43)●	50.748% (0.42)●	49.424% (0.50)●
SPLOT-Generated-FM-1000-3	2	96.880% (0.18)	97.051% (0.16)○	96.948% (0.16)‡	96.965% (0.19)‡
	3	90.563% (0.28)	90.153% (0.28)●	90.117% (0.28)●	89.675% (0.30)●
	4	79.795% (0.37)	78.733% (0.33)●	78.835% (0.31)●	77.803% (0.34)●
	5	65.816% (0.34)	64.185% (0.43)●	64.430% (0.45)●	63.101% (0.43)●
	6	51.124% (0.36)	49.831% (0.44)●	49.969% (0.33)●	48.804% (0.32)●
SPLOT-Generated-FM-1000-4	2	99.542% (0.04)	99.470% (0.08)●	99.295% (0.06)●	99.374% (0.10)●
	3	96.600% (0.19)	96.309% (0.22)●	95.746% (0.21)●	95.880% (0.26)●
	4	89.158% (0.20)	88.278% (0.27)●	87.751% (0.25)●	87.600% (0.35)●
	5	75.672% (0.34)	74.888% (0.37)●	74.117% (0.31)●	73.766% (0.42)●
	6	59.334% (0.32)	58.243% (0.41)●	57.958% (0.38)●	57.361% (0.46)●
SPLOT-Generated-FM-1000-5	2	98.688% (0.05)	98.460% (0.13)●	98.405% (0.07)●	98.189% (0.19)●
	3	93.497% (0.20)	92.883% (0.22)●	92.813% (0.21)●	92.186% (0.41)●
	4	83.048% (0.31)	81.908% (0.38)●	81.880% (0.37)●	80.848% (0.45)●
	5	66.994% (0.32)	65.562% (0.31)●	65.596% (0.34)●	64.400% (0.44)●
	6	50.548% (0.27)	49.238% (0.46)●	49.285% (0.37)●	48.332% (0.34)●
SPLOT-Generated-FM-1000-6	2	99.322% (0.07)	99.130% (0.09)●	99.178% (0.07)●	98.949% (0.13)●
	3	95.717% (0.19)	94.993% (0.23)●	95.203% (0.17)●	94.480% (0.23)●
	4	86.791% (0.31)	85.454% (0.35)●	85.802% (0.26)●	84.374% (0.29)●
	5	72.524% (0.38)	70.676% (0.44)●	71.082% (0.40)●	69.634% (0.39)●
	6	55.533% (0.37)	53.638% (0.38)●	54.145% (0.33)●	52.811% (0.43)●
SPLOT-Generated-FM-1000-7	2	99.444% (0.04)	99.383% (0.06)●	98.784% (0.21)●	99.300% (0.08)●
	3	96.230% (0.13)	95.934% (0.18)●	94.686% (0.31)●	95.534% (0.23)●
	4	88.228% (0.23)	87.377% (0.30)●	86.175% (0.38)●	86.787% (0.33)●
	5	75.439% (0.31)	74.381% (0.46)●	73.180% (0.45)●	73.499% (0.39)●
	6	58.977% (0.43)	57.810% (0.36)●	57.105% (0.37)●	57.318% (0.41)●
SPLOT-Generated-FM-1000-8	2	97.391% (0.16)	97.333% (0.17)‡	97.200% (0.15)●	97.160% (0.17)●
	3	92.253% (0.20)	91.665% (0.30)●	91.471% (0.27)●	91.197% (0.31)●
	4	83.669% (0.30)	82.580% (0.27)●	82.482% (0.35)●	81.668% (0.40)●
	5	71.739% (0.30)	70.348% (0.42)●	70.192% (0.34)●	68.957% (0.40)●
	6	58.611% (0.32)	56.961% (0.41)●	56.848% (0.33)●	55.763% (0.38)●
SPLOT-Generated-FM-1000-9	2	99.137% (0.09)	99.031% (0.10)●	99.070% (0.09)●	98.974% (0.09)●
	3	96.099% (0.17)	95.647% (0.17)●	95.628% (0.20)●	95.281% (0.21)●
	4	89.783% (0.26)	88.846% (0.25)●	88.765% (0.29)●	88.053% (0.27)●
	5	79.166% (0.31)	78.033% (0.27)●	77.879% (0.41)●	76.951% (0.44)●
	6	65.684% (0.37)	64.465% (0.37)●	64.081% (0.50)●	63.349% (0.46)●
SPLOT-Generated-FM-1000-10	2	99.208% (0.05)	99.081% (0.10)●	98.981% (0.08)●	98.892% (0.13)●
	3	95.398% (0.22)	94.852% (0.24)●	94.736% (0.19)●	94.373% (0.28)●
	4	85.858% (0.26)	84.956% (0.33)●	84.747% (0.30)●	83.989% (0.45)●
	5	71.810% (0.29)	70.496% (0.36)●	70.349% (0.36)●	69.301% (0.42)●
	6	55.478% (0.42)	54.211% (0.39)●	54.202% (0.34)●	53.173% (0.42)●
# Best or second best		64/65	47/65	19/65	2/65

small, moderate and large FMs. As shown in Fig. 1, the t-wise coverage for NS is better than (or at least comparable to) that for GA throughout the search process on all the FMs selected regardless of the value of t. In particular,

TABLE 7
Mean values and standard deviations (in brackets) of the t-wise coverage on large-scale FMs for $t = 2, \ldots, 6$. The best and the second-best results are indicated by a dark-gray and a light-gray background, respectively.

Feature model	t-wise	NS	GA	SamplingDown	Unpredictable
ecos-icse11	2	99.719% (0.06)	99.666% (0.06)•	98.529% (0.09)•	99.449% (0.19)•
	3	98.079% (0.12)	97.451% (0.15)•	95.160% (0.29)•	96.578% (0.54)•
	4	93.057% (0.19)	90.984% (0.29)•	88.167% (0.38)•	89.851% (0.57)•
	5	83.807% (0.31)	80.289% (0.32)•	78.055% (0.45)•	79.371% (0.65)•
	6	69.969% (0.33)	64.636% (0.31)•	64.611% (0.58)•	64.757% (0.59)•
freebsd-icse11	2	99.589% (0.06)	99.463% (0.10)•	99.311% (0.09)•	92.629% (0.11)•
	3	98.187% (0.11)	98.028% (0.13)•	97.221% (0.19)•	87.121% (0.13)•
	4	93.754% (0.19)	93.796% (0.19)‡	91.965% (0.22)•	78.976% (0.25)•
	5	84.087% (0.40)	84.112% (0.22)‡	82.111% (0.27)•	67.167% (0.41)•
	6	67.631% (0.47)	67.820% (0.29)‡	65.617% (0.45)•	51.573% (0.36)•
Automotive01	2	96.384% (0.09)	96.304% (0.14)‡	95.604% (0.39)•	96.203% (0.21)•
	3	89.067% (0.20)	88.879% (0.24)‡	87.503% (0.45)•	88.538% (0.20)•
	4	79.327% (0.30)	79.021% (0.23)•	77.312% (0.40)•	78.380% (0.25)•
	5	65.131% (0.26)	64.309% (0.30)•	63.100% (0.38)•	63.868% (0.27)•
	6	51.664% (0.24)	51.090% (0.20)•	49.864% (0.44)•	49.911% (0.30)•
SPLOT-Generated-FM-5000	2	98.291% (0.12)	98.299% (0.17)‡	98.039% (0.13)•	97.956% (0.22)•
	3	92.215% (0.26)	92.197% (0.22)‡	91.493% (0.26)•	91.205% (0.28)•
	4	80.949% (0.28)	80.549% (0.34)•	79.605% (0.30)•	79.133% (0.39)•
	5	65.854% (0.44)	65.263% (0.17)•	64.430% (0.31)•	63.715% (0.29)•
	6	49.443% (0.25)	48.880% (0.29)•	48.152% (0.43)•	47.813% (0.42)•
2.6.28.6-icse11	2	98.676% (0.11)	98.797% (0.13)○	97.923% (0.17)•	97.293% (0.20)•
	3	95.744% (0.12)	96.153% (0.19)○	94.769% (0.30)•	94.152% (0.31)•
	4	90.044% (0.24)	90.351% (0.30)○	88.694% (0.26)•	87.247% (0.34)•
	5	79.877% (0.17)	79.773% (0.36)‡	78.564% (0.30)•	76.633% (0.27)•
	6	63.741% (0.24)	63.286% (0.34)•	62.410% (0.31)•	60.358% (0.33)•
Automotive02_V3	2	78.260% (0.32)	78.075% (0.40)‡	78.128% (0.20)‡	78.191% (0.33)‡
	3	59.576% (0.35)	59.675% (0.23)‡	59.540% (0.34)‡	59.487% (0.20)‡
	4	43.145% (0.36)	43.196% (0.26)‡	42.900% (0.28)‡	43.011% (0.36)‡
	5	30.225% (0.31)	30.417% (0.22)‡	30.323% (0.31)‡	30.283% (0.21)‡
	6	20.713% (0.24)	20.808% (0.16)‡	20.705% (0.24)‡	19.943% (0.19)•
# Best or second best		29/30	28/30	1/30	2/30

TABLE 8
Summary of the Mann-Whitney U test results on all FMs

NS v.s.	GA	SamplingDown	Unpredictable
•	$119/155 \approx 77\%$	$138/155 \approx 89\%$	$145/155 \approx 94\%$
‡	$28/155 \approx 18\%$	$15/155 \approx 10\%$	$10/155 \approx 6\%$
○	$8/155 \approx 5\%$	$2/155 \approx 1\%$	$0/155 \approx 0\%$

NS shows an obviously larger t-wise coverage than GA in most cases, and a close t-wise coverage to GA on several large FMs, e.g., Automotive01, SPLOT-Generated-FM-5000 and Automotive02. As observed, the t-wise coverage, in the majority of the cases, tends to increase as the search proceeds. However, we notice that the curve of the t-wise coverage for GA obviously declines on CounterStrikeSimpleFeatureModel for $t = 5$ and $t = 6$, and ecos-icse11 for $t = 6$. This fact implies that the search via GA may hamper the t-wise coverage in some cases. One of the most likely explanations of this phenomenon is that the fitness function adopted by GA gives an improper measure of the similarity [7], [9].

Finally, notice that the time budget for testing is usually limited in practice [4], [9]. Therefore, products should be prioritized such that the t-wise coverage provided by these configurations is achieved faster. That way, it potentially increases the probability of improving the fault detection ratio [7]. How do the four algorithms perform in terms of prioritizing products? To answer this question, we plot in Fig. 2 the average t-wise ($t = 6$) coverage with respect to the number of used configurations on nine representative FMs. For instance, regarding SmartHomev2.2, NS enables covering around 70 percent of the 6-sets on average (over 30 runs) with 50 percent of the configurations. According to [4], the area under the curve reflects the effectiveness of the prioritization. The larger this area, the better the prioritization. As observed, NS performs either (significantly) better than or similarly to its competitors on these FMs. It must be noted that both GA and SamplingDown use a similarity-based prioritization approach to order products. In contrast, our NS algorithm does not explicitly consider prioritization (because we mainly focus on product sampling in this paper. Certainly, any prioritization approach can be applied to the resulting samples). Despite of this, NS can still be effective concerning prioritization. One of the most likely explanation is that configurations generated by NS are already with high quality. Even with the default order, they can achieve large t-wise coverage as early as possible.

4.2.4 *RQ2 summary*

The following are some conclusions drawn from the above numerical results.

- Generally, the two search-based approaches, i.e., NS and GA, are more effective than the two sample-based approaches, i.e., SamplingDown and Unpredictable. This observation can be explained by the results of the correlation analysis performed in Section 4.1. It has been shown that similarity metrics used in NS and GA have a significantly positive correlation with the t-wise coverage in most cases. Therefore, it is not surprising that the two search-based approaches

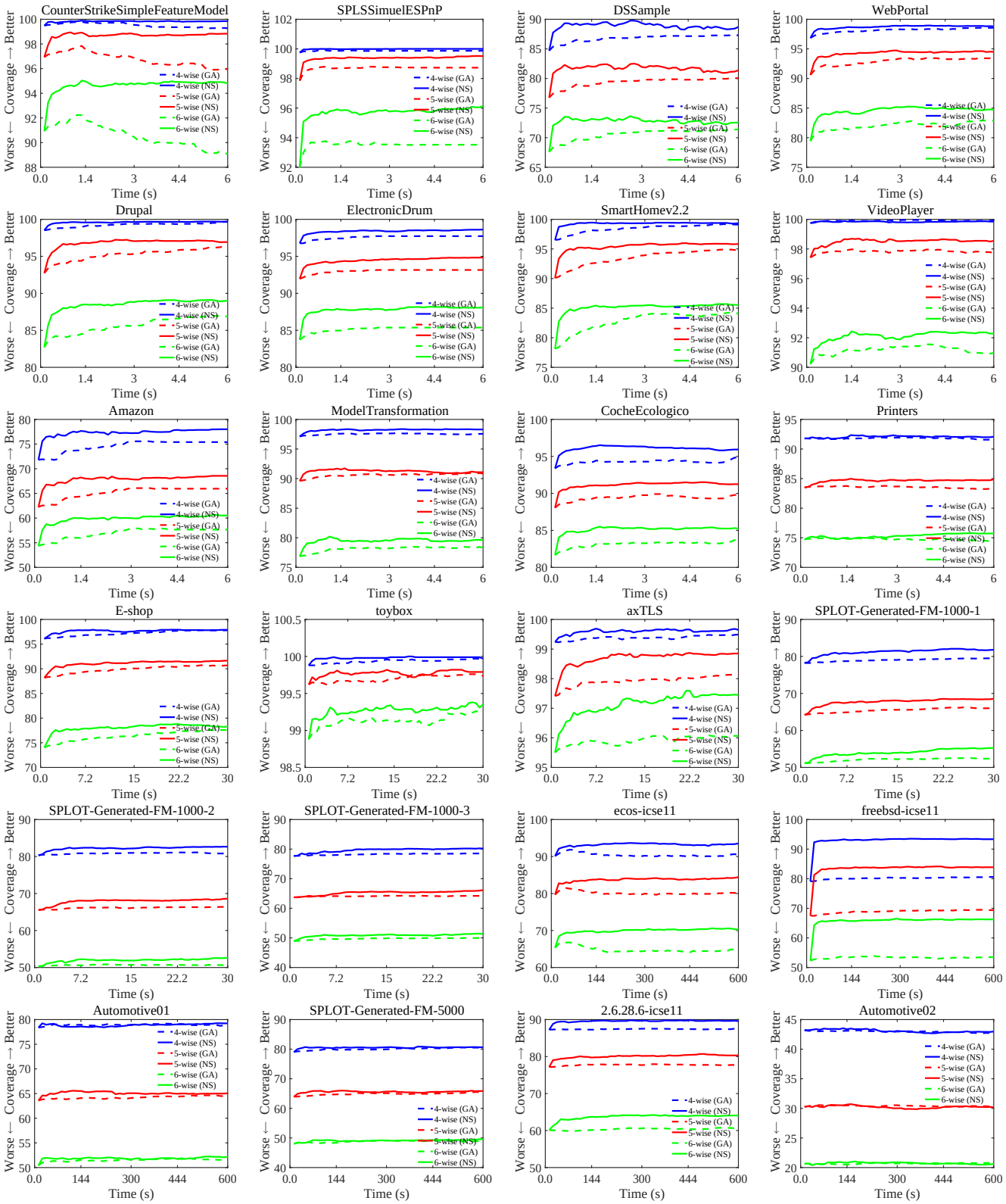


Fig. 1. The t-wise coverage obtained by NS and GA for $t = 4, 5, 6$ in a typical run during the search process. Both algorithms start with the same initial population.

are better than sample-based approaches, because the improvement on similarity metrics will potentially increase the t-wise coverage. In contrast, the two sample-based algorithms generate a certain number of configurations in an unpredictable way, without exploring sufficient configurations. As a result, there is no enough selection pressure towards configurations that can contribute to the t-wise coverage.

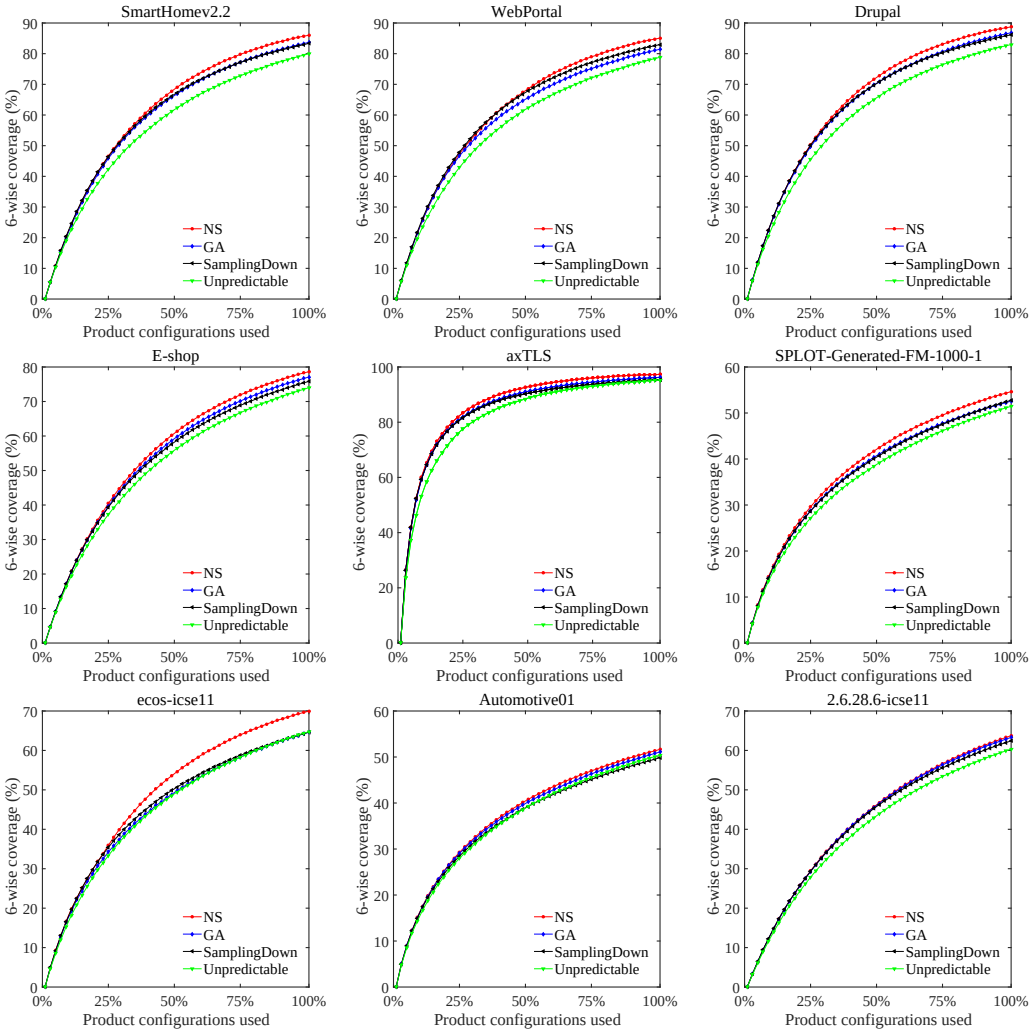


Fig. 2. Comparisons of prioritization on nine representative FMs. The larger the area under the curve, the better the prioritization.

- In general, NS outperforms GA concerning both the final t-wise coverage and the performance over time. This observation is in line with the finding of the correlation analysis. It has been observed in Section 4.1 that the novelty score used in NS has a (much) stronger positive correlation with the t-wise coverage than the similarity-based fitness used in GA. This can be the primary reason for the superiority of NS over GA.
- Concerning the prioritization capacity, NS is promising even without employing any prioritization strategy, being even more effective than those algorithms in which prioritization is explicitly considered. This indicates the potential of NS in improving the fault detection ratio.

4.3 Comparison of different ways of generating new configurations (RQ3)

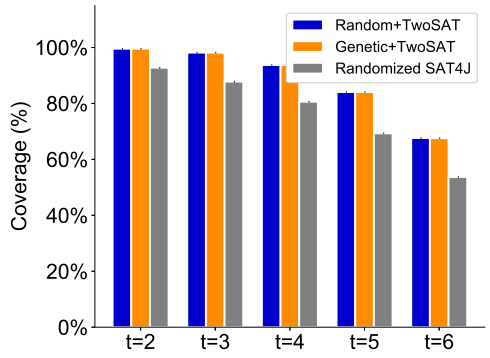
This part investigates how the performance of NS is affected by the ways (or strategies) in which new configurations are generated. The way of generating new individuals is an important aspect of a search-based approach. Currently, the unpredictable way, realized by the randomized SAT4J solver, forms a state-of-the-art approach. It remains open whether this approach can be further improved by using two types of solvers as in [59], [60], and whether it is necessary to introduce traditional genetic operators. In this section, the answers to these questions will be made clear after a comparative study.

4.3.1 *Experiment setup and results*

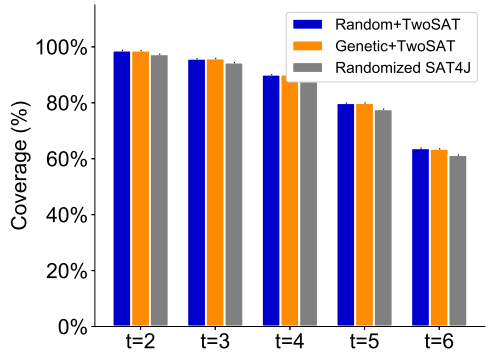
We empirically evaluate the following three strategies of generating new configurations. The first one, as described in Section 3.2 and currently being used, generates each time a random configuration that is to be handled by two different types of SAT solvers in case of infeasibility: probSAT [63] for repair and randomized SAT4J for replacement. In the remainder of this paper, we refer to this strategy as *Random+TwoSAT*. The second strategy follows a traditional way as in evolutionary algorithms. In this strategy, genetic operators (uniform crossover and single-point mutation) are adopted to generate a new configuration. As in the first strategy, this configuration will be then handled by two SAT solvers. We refer to this strategy as *Genetic+TwoSAT* hereafter. Comparisons with this strategy allow us exploring how genetic operators perform compared with the simple

randomization. Finally, the third strategy relies solely on the randomized SAT4J to generate unpredictable configurations [4]. We refer to this strategy as *Randomized SAT4J*. Notice that requesting to a randomized SAT4J solver to get unpredictable configurations forms a state-of-the-art approach in the SPL engineering [8], [57], [58], [59], [60], [61]. Choosing this strategy as a baseline enables us to investigate whether it is necessary to employ two different SAT solvers.

Considering Table 9, the differences among the three strategies are not statistically significant on 4 out of the 6 large-scale FMs regardless of the value of t. As observed, however, both *Random+TwoSAT* and *Genetic+TwoSAT* significantly outperform *Randomized SAT4J* on freebsd-icse11 and 2.6.28.6-icse11 with respect to all the values of t. Fig. 3 illustrates this behavior in the form of histogram. As seen, the t-wise coverage obtained by the first two strategies are obviously higher than that obtained by *Randomized SAT4J*, especially on freebsd-icse11. By comparing *Random+TwoSAT* and *Genetic+TwoSAT* in Table 9, one can find that the genetic operators perform equally to the simple randomization method in generating new configurations.



(a) freebsd-icse11



(b) 2.6.28.6-icse11

Fig. 3. The t-wise ($t = 2, \ldots, 6$) coverage obtained by the there configuration generation strategies on (a) freebsd-icse11, and (b) 2.6.28.6-icse11.

4.3.2 RQ3 summary

Experiments performed in this section emphasize that different ways of generating new configurations have similar performance in most cases, but can indeed make a difference in some cases. In particular, *the use of two SAT solvers should be encouraged as it can achieve (much) better or at least comparable performance to the state-of-the-art strategy where only one SAT solver is employed*. This conclusion is in line with the one drawn in [59]. It is stated that the simultaneous use of two types of SAT solvers is more effective than the use of only one in the context of configuring SPLs [59]. In this paper, we show, for the first time, this is also the case for the similarity-based testing of SPLs. Moreover, *when aided by two SAT solvers, ways of generating new configurations make no difference*. Indeed, the simple randomization method can be as effective as genetic operators. The former, however, is quite easy to use as it requires no efforts of tuning control parameters, like the crossover and mutation probabilities involved in genetic operators. Therefore, its use is advisable.

4.4 Parameter study (RQ4)

From the practical point of view, it is important to provide some suggestions on the setting of the key parameters involved in the NS algorithm. As shown in Algorithm 1, the repair probability (P_r) and the neighbor size (N_b) are two key parameters that needs to be carefully tuned.

4.4.1 Sensitivity analysis on P_r

The $P_r \in [0, 1]$ determines the probability of using probSAT solver, and its impact on the t-wise coverage is investigated by examining some typical values. To this end, we change P_r from 0.0 to 1.0 with a step size 0.1, and present the t-wise coverage over 30 runs in the form of boxplots for each value of P_r. This way, it enables us to observe the trend of the average t-wise coverage as P_r increases. As observed in Fig 4, the t-wise ($t = 2, \ldots, 6$) coverage slightly fluctuates as P_r increases from 0.0 to 1.0 on the two representative small-scale FMs (i.e., WebPortal and CocheEcologico), and tends to descend on the two moderate FMs (i.e., E-shop and axTLS), especially when $P_r \geq 0.5$. This observation suggests that the value of P_r should not be too large. Considering the two large FMs (i.e., freebsd-icse11 and 2.6.28.6-icse11), the t-wise coverage is improved obviously when P_r is changed from 0.0 to 0.1, and retains relatively stable for $P_r \geq 0.1$. This implies that the value of P_r should not be too small (< 0.1), either. Considering all the above scenarios, it is reasonable to restrict P_r to the region $[0.1, 0.5]$ for a common usage.

4.4.2 Sensitivity analysis on N_b

In a similar way, we conduct sensitivity analysis experiments on the parameter N_b. We change N_b from 10 to 100 with a step size 10. In addition, $N_b = 2$ is also examined. In this case, the novelty score of a solution is determined by its two closest neighbors (including itself). For each FM and each value of N_b, the t-wise coverage over 30 runs is exhibited in the form of boxplots. Moreover, we use a green line to connect two adjacent average values. By doing so, it makes easy observing how the average t-wise coverage changes.

We can identify the following three types of curves from Fig. 5.

- Type-I curve, as observed on FMs from CounterStrikeSimpleFM to WebPortal (note that the order is from left to right, and from top to bottom), rises as N_b increases from 2, and quickly reaches a peak around $N_b = 10$ or 20. It then declines until N_b reaches 70

TABLE 9
Comparison of different ways of generating new configurations on large-scale FMs

Feature model	t-wise	*Random+TwoSAT*	*Genetic+TwoSAT*	*Randomized SAT4J*
ecos-icse11	2	99.719% (0.06)	99.696% (0.04)‡	99.715% (0.04)‡
	3	98.079% (0.12)	98.101% (0.13)‡	98.048% (0.18)‡
	4	93.057% (0.19)	93.111% (0.24)‡	93.137% (0.28)‡
	5	83.807% (0.31)	83.827% (0.22)‡	83.789% (0.29)‡
	6	69.969% (0.33)	69.789% (0.35)‡	69.760% (0.37)‡
freebsd-icse11	2	99.589% (0.06)	99.592% (0.05)‡	92.801% (0.07)•
	3	98.187% (0.11)	98.172% (0.10)‡	87.818% (0.10)•
	4	93.754% (0.19)	93.725% (0.22)‡	80.627% (0.20)•
	5	84.087% (0.40)	84.077% (0.39)‡	69.320% (0.23)•
	6	67.631% (0.47)	67.585% (0.37)‡	53.683% (0.33)•
Automotive01	2	96.384% (0.09)	96.439% (0.19)‡	96.349% (0.10)‡
	3	89.067% (0.20)	88.974% (0.30)‡	88.955% (0.23)‡
	4	79.327% (0.30)	79.280% (0.36)‡	79.125% (0.25)‡
	5	65.131% (0.26)	65.125% (0.28)‡	65.129% (0.26)‡
	6	51.664% (0.24)	51.898% (0.33)‡	51.715% (0.31)‡
SPLOT-Generated-FM-5000	2	98.291% (0.12)	98.301% (0.12)‡	98.249% (0.13)‡
	3	92.215% (0.26)	92.247% (0.18)‡	92.355% (0.18)‡
	4	80.949% (0.28)	80.817% (0.31)‡	80.932% (0.31)‡
	5	65.854% (0.44)	65.711% (0.18)‡	65.694% (0.29)‡
	6	49.443% (0.25)	49.435% (0.30)‡	49.479% (0.41)‡
2.6.28.6-icse11	2	98.676% (0.11)	98.645% (0.10)‡	97.359% (0.11)•
	3	95.744% (0.12)	95.848% (0.20)‡	94.381% (0.16)•
	4	90.044% (0.24)	90.105% (0.20)‡	87.678% (0.20)•
	5	79.877% (0.17)	79.910% (0.30)‡	77.686% (0.33)•
	6	63.741% (0.24)	63.551% (0.32)‡	61.317% (0.37)•
Automotive02_V3	2	78.260% (0.32)	78.294% (0.24)‡	78.331% (0.28)‡
	3	59.576% (0.35)	59.474% (0.35)‡	59.503% (0.31)‡
	4	43.145% (0.36)	43.155% (0.28)‡	43.147% (0.30)‡
	5	30.225% (0.31)	30.573% (0.17)‡	30.387% (0.22)‡
	6	20.713% (0.24)	20.638% (0.21)‡	20.625% (0.24)‡

or 80. Finally, the curve rises, in general, again till $N_b = 100$. Note that the average t-wise coverage obtained on axTLS and WebPortal roughly follows the above curve, rather than exactly. According to Table 1, the median value of the number of free features for the FMs in this group is 37.5.

- Type-II curve, as observed on FMs from Drupal to ecos-icse11, is unimodal. The only peak is obtained when N_b falls in the region $[30, 50]$. Note that the median value of the number of free features for these FMs is 76.
- Type-III curve, as observed on FMs from E-shop to 2.6.28.6-icse11, tends to increase along with the value of N_b until it reaches around 80, and then the curve slightly drops till to the end (note that 2.6.28.6-icse11 is an exception, in which the curve always increases). The median value of the number of free features in these group is 2218.

According to the above observations, we are aware that the optimal value of N_b seems to be relevant to the model size, more precisely the number of free features. In general, higher value of N_b is required to obtain decent performance for larger feature models. This phenomenon can be explained as follows. As the number of (free) features increases, the size of the search space grows exponentially. It naturally follows that more points are need to provide a more precise estimation to the density around a point. Therefore, the number of neighbors should be increased in this case.

4.4.3 RQ4 summary

The parameter study performed brings out the following outcomes. First, *the NS algorithm prefers to relatively small P_r*. In other words, less calls should be given to the probSAT than to the randomized SAT4J. Generally, the algorithm is not sensitive to the value of $P_r \in [0.1, 0.5]$. Second, *the (nearly) optimal value of the parameter N_b varies with the size of the FMs*. It is found that higher N_b is required for larger models. This is a practical guidance when using NS in the context of SPL testing.

5 Threats to Validity

In this section we briefly discuss threats to validity and how they could be mitigated. We consider threats to internal validity, external validity and construct validity.

Internal validity. This is concerned with any aspect which may lead to bias. Potential errors in the implementation of our algorithm and the tools used for comparison could affect the presented results and lead to this type of threats. To diminish these threats, we carefully tested our implementation of NS by performing unit testing, and by analyzing the outcomes step by step on small FMs. The comparison with existing tools gave us confidence in our implementation. For the algorithms used for comparison, they are implemented by the code provided by their authors. In addition, we make our implementation and experiments data publicly available[10] to enable reproducibility and to reduce the aforementioned threats.

10. https://www.zenodo.org/deposit/3928922

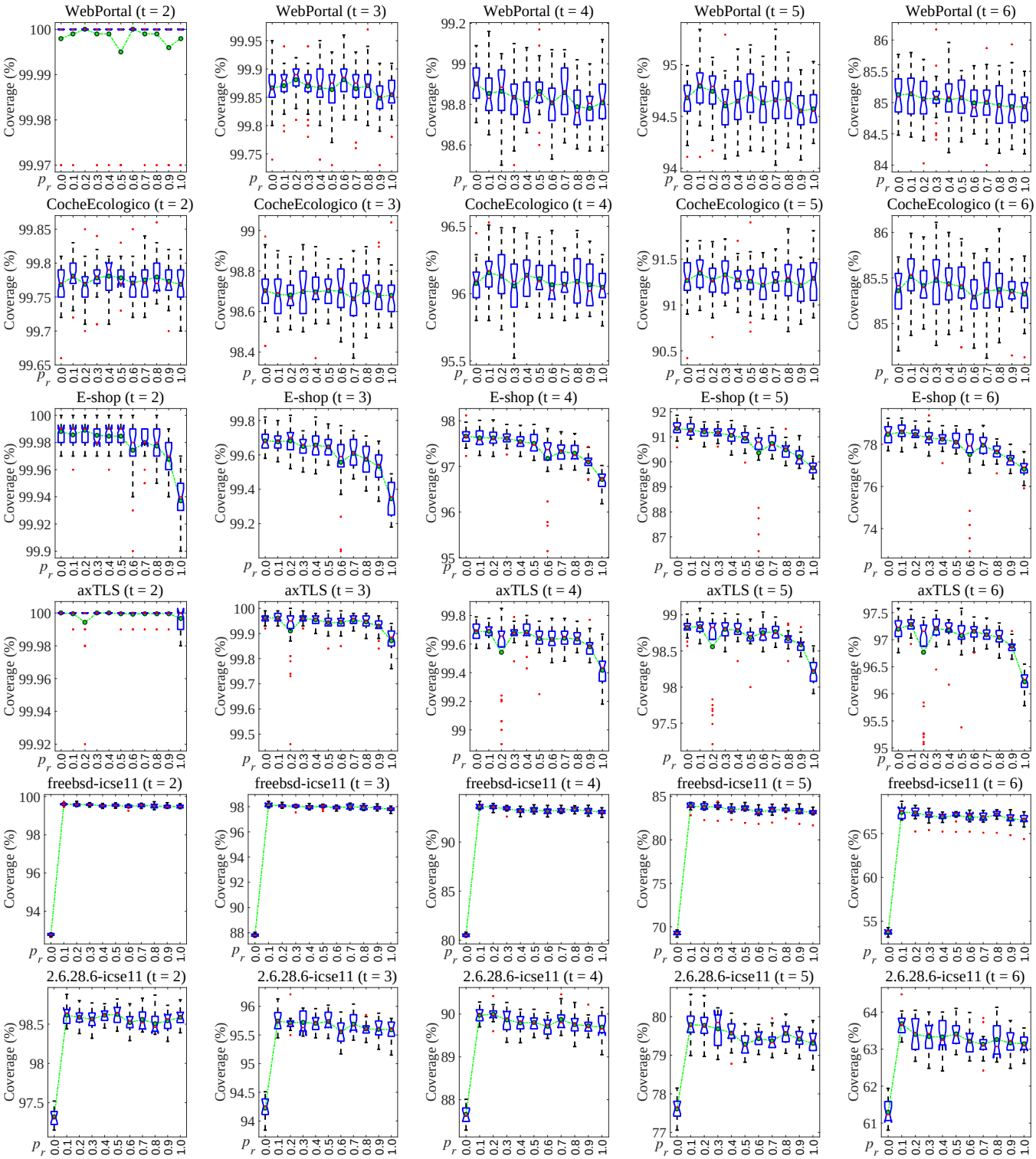


Fig. 4. Parameter study of P_r on six representative small-, moderate- and large-scale FMs for $t = 2, \ldots, 6$

Due to the stochastic nature of the techniques under study, the outcome of a run can be different from the next runs. To reduce risks caused by random effects, we independently run each of the algorithms 30 times, and compare the performance based on means and standard deviations. In addition, to make comparisons reliable, nonparametric statistical tests are applied.

Internal validity threats may be also introduced in the computation of the t-wise coverage. Indeed, as discussed previously, it is different or even practically impossible to compute all the valid t-sets for large FMs and high t values. Therefore, valid t-sets used to compute the coverage are sampled. This means that the presented t-wise coverage is not real, except on small and moderate FMs for $t = 2$ (exact valid t-sets are used in this case). To mitigate these threats, we use the same set of valid t-sets (either sampled

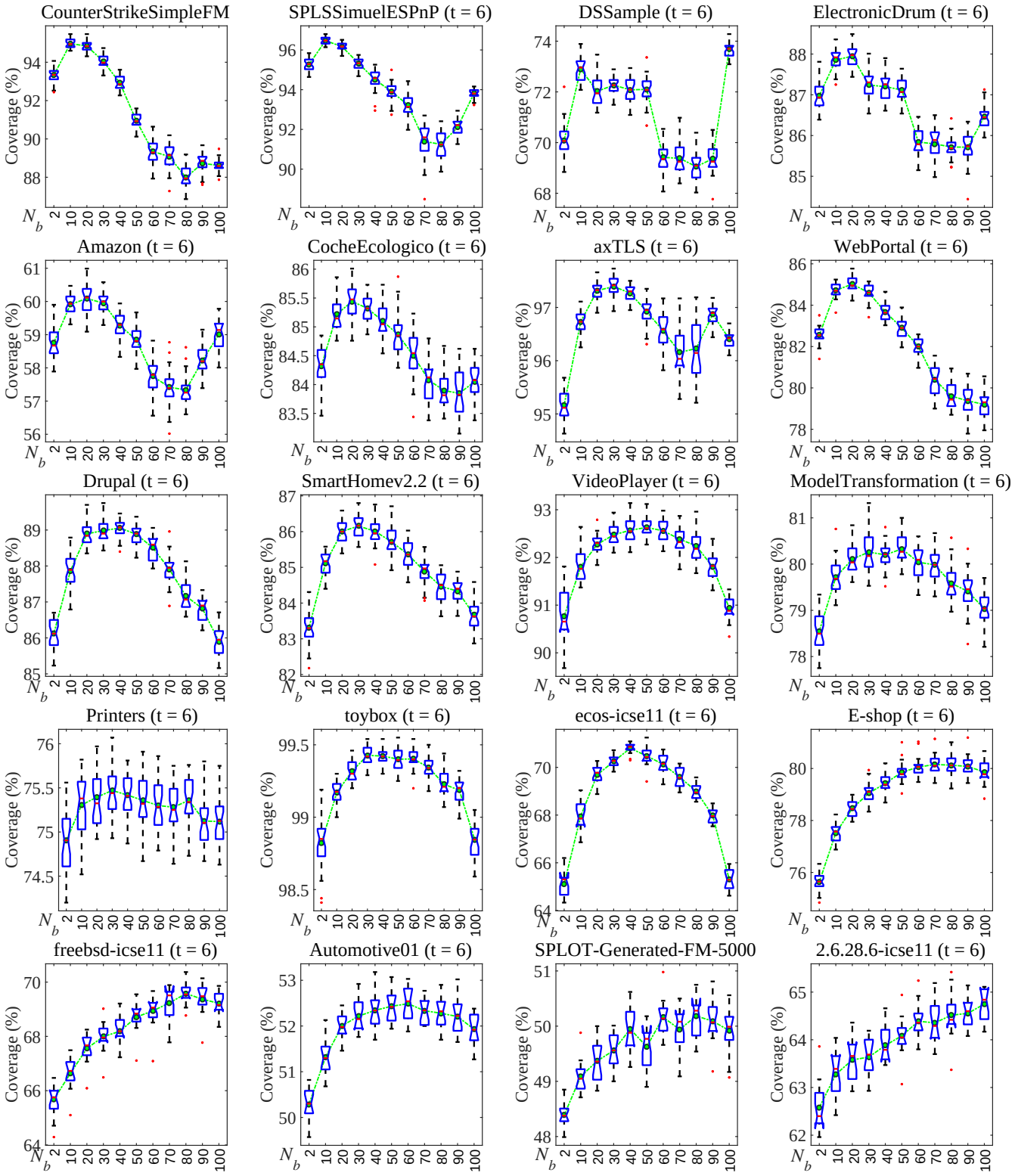


Fig. 5. Parameter study of N_b on 20 representative FMs for $t = 6$.

or exactly generated) to compute the t-wise coverage of the configurations returned by each algorithm. This way, it enables a relative performance comparison among the algorithms.

External validity. This threat commonly exists in software engineering research, and it is related to the degree to which we can generalize from the experiments. Indeed, we cannot guarantee that the proposed algorithms will provide similar results on different sets of FMs. This threat comes from the size of the feature models, as well as the size of constraints. To mitigate this threat, we use a relatively large set of 31 FMs with the size ranging from 24 to 18,434. Also,

the number of constraints varies from the smallest 35 to the largest 347,557. In addition, both realistic and randomly generated feature models are carefully selected to provide a variety of situations.

Construct validity. We choose $t = 2, \ldots, 6$ in the empirical study. This is in line with the previous studies. The pairwise coverage ($t = 2$) is a widely used test criterion in SPL testing. According to [33], [34], [35], there is also a practical need to deal with high interaction strengths ($t > 2$). Therefore, choosing $t = 2, \ldots, 6$ is of interest in practice.

6 Related Work

In recent years, there has been a growing interest in SPL testing, and this is reflected by several surveys and systematic mapping studies on this topic [13], [77], [78], [79], [80], [81], [82], [83]. Some of these surveys and mapping studies cover various factors and aspects of the SPL testing, e.g., [77], [78], [79], [82], while some of them focus on a particular aspect of the SPL testing, e.g., CIT-based approaches [13], [81], EC-based approaches [80] and test coverage criteria [82]. In this work, we focus on product sampling [10] for SPL testing. In Section 6.1, we briefly review some prominent t-wise sampling techniques. Since these techniques can hardly scale to large product lines and/or high interaction strengths, similarity is used as a surrogate metric for the t-wise coverage, making similarity-based SPL testing popular in this domain. Related similarity-based testing approaches are summarized in Section 6.2.

6.1 T-wise product sampling

The Chvatal [18] is greedy algorithm originally proposed to deal with the minimum set-covering problem, an NP-complete problem. Johansen et al. [12] adapted and improved this algorithm to approximate optimal solutions for the minimal covering array. A t-wise covering array is a subset of products that covers all the valid t-sets [15]. A covering array is typically represented by a table with each row representing a feature and each column representing a product. With Chvatal, configurations are added in an incremental manner until all the valid t-wise feature combinations are covered at least once. More specifically, the algorithm begins with generating all the t-wise feature combinations, denoted by U. Then, an empty configuration is created, and these feature combinations are added into this configuration one by one. Each time a new combination is added, and the resulting configuration is checked by an SAT solver. If the configuration is valid and contains at least one uncovered combination, then it is added to the sample set. At the same time, this newly added combination is removed from U. Otherwise, the combination is removed from the configuration. The creation of configurations continues until all the valid combinations are covered. Built upon the Chvatal algorithm [12] with several enhancements, such as identifying invalid feature combinations early, and parallelizing the sampling process, the ICPL [19] algorithm is able to generate covering arrays for large-scale feature models but it is limited to $t = 3$. For the largest 2.6.28.6-icse11 model, ICPL is only capable of generating 1-wise and 2-wise covering arrays.

Oster et al. [16] proposed the MoSo-PoLiTe (**Mo**del-based **So**ftware **Pro**duct **L**ine **Te**sting) framework to generate a minimal set of products covering all pairs of feature combinations. This is achieved by combining binary constraint solving and forward checking [84]. The algorithm starts with the first pair of features and iteratively adds the rest ones by applying a forward checking to determine whether the selected pair can be combined with the remaining pairs to create a valid product. Furthermore, MoSo-PoLiTe allows including pre-defined products as part of the covering arrays. Experiments were conducted on nine realistic FMs, with the largest one containing 287 features. Similarly, Hervieu et al. [85] adopted constraint programming to generate minimal 2-wise covering arrays for SPLs.

The **In**cremental samp**Ling** (IncLing), proposed by Al-Hajjaji et al. [21], follows the general concept of ICPL [19], but incorporates several crucial modifications to improve the performance, such as detecting invalid combinations at the beginning, using feature ranking heuristic, and detecting conditionally dead or core features. With IncLing, the algorithm takes into account already generated and tested products when selecting the next product to be added into the sample. The next product is selected in a greedy manner to cover uncovered feature combinations as many as possible. Moreover, to increase the diversity among products, the algorithm chooses dissimilar pair-wise feature combinations when a new product is generated each time. This way, it potentially increases the possibility of fault detection. Experiment results on a corpus of real-world and artificial feature models demonstrated that IncLing is more efficient than existing sampling algorithms, e.g., Chvatal [12] and ICPL [19]. Although large models (including 2.6.28.6-icse11) are used in their empirical evaluations, the strength is limited to $t = 2$.

Perrouin et al. [66] proposed a tool for generating t-wise samples from a feature diagram. Using the divide and conquer technique, the problem of t-wise sampling is divided into several sub-problems that are solved automatically. To do so, a given feature diagram and its interactions should be first transformed into a set of constraints into Alloy, a formal modeling language. Then, a set of products are generated using the resulting model, which provide t-wise coverage. The evaluation of this tool is performed on a real-world feature model called AspectOPTIMA.

Garvin et al. [17] used simulated annealing to construct CIT samples of SPLs. The algorithm, named CASA, consists of two parts: outer search and inner search. The outer search, implemented by a binary search procedure, is concerned with the minimization of the sample size, while the inner search, implemented by the simulated annealing, aims at finding a covering array. To this end, one value of the covering array is changed each time and this change is guided by the number of t-sets that are not covered. A SAT solver is used to control whether the change should be accepted or rejected. Moreover, several strategies were incorporated to improve the performance, and experimental results confirmed the usefulness of these strategies. According to [17], CASA can handle relatively high strengths, with up to $t = 4$ for FMs with no more than 52 features. For models beyond this size, the algorithm timed out after 27 days when $t = 4$.

Ensan et al. [22] devised, on the basis of genetic algorithms, an approach to generate test suites for SPLs. This algorithm starts with a set of randomly generated configurations, and gradually improves the quality of the test suite by iteratively applying genetic operators, i.e., crossover, mutation and natural selection. As opposed to the aforementioned approaches, Ensan's approach does not guarantee a full t-wise coverage (only $t = 1$ is considered in [22]) because the fitness function used indirectly measures coverage. It should be mentioned that new individuals developed based on crossover and mutation are not necessarily valid. In this case, invalid ones are discarded and are not included as a part of the new generation. This may lead to scalability issues (according to [4], Ensan's approach does not scale over 300 features). Notice that, in addition to the t-wise coverage, code coverage [86] and mutation scores [87] are also considered as a criterion when applying genetic algorithms to the product sampling process.

Marijan et al. [23] proposed a pairwise testing framework for SPLs. It integrates feature modelling, combinatorial interaction testing, and constraint programming techniques. In particular, the framework first, extracts variability in an SPL as a feature model, and then employs an optimization algorithm to generate a minimal set of valid products covering all pairwise feature combinations for a given time interval. The optimization algorithm is implemented by constraint programming techniques, including a dedicated branch-and-bound algorithm. This proposal was evaluated on an industrial SPL with 78 features. The experiments reveal that this framework can reduce the sample size while guaranteeing a full pairwise coverage at the same time.

Cmyrev and Reissing [24] suggested a new test approach to generate a test suite that is as small as possible, and yields a full requirements coverage and a full feature coverage ($t = 1$). Since finding the optimal set of test cases is very hard, a greedy algorithm and a simulated annealing approach are explored to find the nearly optimal set in a fast way. Experiments on two case studies (with at most 37 features) in the automotive industry showed that the greedy algorithm is superior to the simulated annealing concerning both effectiveness and efficiency.

According to [28], most of the aforementioned t-wise sampling techniques do not scale to large product lines and/or high interaction strengths. Indeed, small-sized FMs are used in most of the above prior works, considering often $t = 1$ or $t = 2$. As discussed in Section 1, there is a practical need of dealing with larger FMs and higher strengths. To alleviate the limitation of existing t-wise sampling techniques, several approaches have been proposed to reduce the configuration space [88], [89]. These approaches, however, often require more domain knowledge to guide the sampling process. As an alternative to the t-wise sampling, the similarity-based sampling, which aims at generating a set of configurations achieving a degree of t-wise coverage by maximizing diversity among test cases, can bypassing the scalability issues faced by the t-wise sampling techniques. Moreover, it does not require more domain knowledge than the feature model, i.e., knowledge on valid combinations of features [7]. In the following part, we give an overview on several prominent similarity-based SPL testing techniques.

6.2 Similarity-based SPL testing

In the context of software product lines, similarity has been exploited to partially but efficiently mimic the t-wise coverage for the created product samples. Henard et al. [4] considered the similarity between configurations as the fitness function, and used an (1+1) evolutionary algorithm to optimize it. The use of similarity as a surrogate metric for the t-wise coverage enables the algorithm to generate and prioritize test configurations for large SPLs and high strengths. Results in [4] suggest that *two dissimilar configurations are more likely to cover a greater number of valid t-sets than two similar ones*. This approach is supported by a test generation tool, called PLEDGE [61]. As an application of the similarity testing, Henard et al. [90] evaluated the capability of test suites to kill mutants of feature models, i.e., erroneous feature models derived from an original one. Experiments demonstrated that dissimilar tests suites kill (or detect) more mutants than similar ones, thus validating the effectiveness of similarity-based SPL testing. Further, Fischer et al. [40] showed that Henard's similarity approach is useful to detect interaction faults in the Drupal case study in which real fault data are used.

Al-Hajjaji et al. systematically studied similarity-based prioritization in SPL testing in four related papers [7], [9], [43], [44]. Similarity-based prioritization, aims at increasing interaction coverage as fast as possible over time, selects configurations based on their performance with respect to similarity. The similarity (i.e., distance) between configurations in [7], [9] are calculated by taking the deselected features into account, making it different from the way in [4]. Furthermore, the approach proposed by Al-Hajjaji et al. [7], [9] incrementally selects configurations with the maximum of the minimum distances to those already chosen, rather than with the maximum of the sum of distances as in [4]. According to Al-Hajjaji et al. [7], [9], the sum of distances as a selection criterion may be misleading in some cases. In [43], Al-Hajjaji et al. extended this prioritization approach by considering the delta models, which can be used to reason about the similarity of products on the solution-space level. Results showed that prioritizing products based on delta modeling can improve the effectiveness of SPL testing. In a more recent work [44], they discussed how the similarity between products of an SPL can be measured considering different types of information. In particular, they distinguished the input information for similarity calculation into problem-space information (e.g., feature-selection similarity, attributes similarity, instance similarity), and solution-space information (e.g., family models similarity). Besides, they discussed the possibility of combining different types of information to form an overall similarity between two products. Finally, the calculation of similarity in different scenarios is implemented in an industrial tool, called pure::variants.

Devroey et al. [45] assessed the SPL behavioural coverage of configurations sampled by two structural testing approaches, i.e., t-wise testing and similarity testing. To this end, they modelled SPLs in terms of feature diagrams and associated featured transitions systems (FTSs), and then computed state, action and transition coverage for a set of sampled configurations. To obtain samples, tools ICPL (t-

wise) [19] and PLEDGE (similarity-based) [61] were used. In a subsequent study [46], they provided a configurable search-based approach to support single and bi-objective similarity-driven test case selection for behavioural SPLs. Empirical results on four case studies showed the relevance of similarity-based test case generation for behavioural SPL models. Moreover, they examined different types of distances in measuring similarity between test cases, finding that Hamming and Jaccard distances are the most efficient.

Lity et al. [47] proposed a similarity-based approach to reorder products in the context of incremental product line analysis. Most of the aforementioned coverage-driven product selection/prioritization techniques [4], [7], [9], [43], [45] select the most dissimilar product as the next products to be tested/analyzed in order to increase the coverage of feature interactions as early as possible. Different from this, the incremental analysis requires a product order where subsequent products are more similar to each other. This way, it can potentially reduce the overall analysis efforts. Lachmann et al. [48] employed a dissimilarity measure (Jaccard distance) to avoid clustered test case orders. This is motivated by the fact that, in a single system model-based testing, dissimilar test cases detect more faults than similar ones [37]. There have been several approaches exploiting similarity to the source code level, e.g., [91], [92], [93]. As pointed out by Al-Hajjaji et al. [44], adapting these approaches that consider the source code into the context of software product line engineering may enhance the analysis of SPLs, including SPL testing.

Similarity has been successfully applied to mimic coverage criteria (e.g., the t-wise coverage) in an effective and scalable manner in prior works [4], [7], [9], [45], [46]. The suitability of similarity as a surrogate metric for the t-wise coverage is mainly demonstrated by empirical results in an intuitive way. No work, to our best knowledge, has been done to reveal the internal relationship between similarity and the t-wise coverage. In this paper, we perform a correlation analysis to explore this relationship, observing that similarity metrics have a significantly positive correlation with the t-wise coverage. This finding consolidates the foundation of search-based similarity-driven SPL testing because improving similarity metrics (e.g., the one used by [4]) consequentially increases the t-wise coverage. Moreover, GA was extensively used in previous works [4], [40], [46], [61]. In this work, we explore, for the first time, the use of NS in similarity-based SPL testing because NS perfectly matches the testing goal, i.e., searching for a set of diverse configurations. We demonstrate that the novelty score has a (much) stronger positive correlation with the t-wise coverage than the fitness function used in GA [4]. This is in line with our empirical results, showing that NS (significantly) outperforms GA in most cases. In addition, we systematically investigate how the performance of NS is affected by its key parameters. Finally, we offer an alternative approach to generate new configurations in search-based SPL testing. This approach exploits two types of SAT solvers as in [59], [60], and it is found to be more effective than the state-of-the-art strategy [4] where only one SAT solver is used.

7 Conclusion and Future Work

Testing SPLs is crucial to avoid fault propagation cross products, but it is challenging because of a huge number of possible products to be tested and a limited amount of test budgets available in practice. The t-wise combinatorial interaction testing (CIT) has been widely used to drastically reduce the size of a test suite, a set of test cases (i.e., products). However, it can hardly scale to large-size SPLs and high interaction strengths. As an alternative to this testing technique, similarity-based SPL testing has shown a great potential in dealing with the scalability issues. However, the rationale of this testing technique deserves a more rigorous exploration. Moreover, genetic algorithms have been extensively used in search-based similarity-driven testing. It is meaningful to exploit the potential benefits of more suitaible/powerful search algorithms in this context.

In this paper, we focus on sampling a set of products as dissimilar as possible by using a search algorithm, called NS. We show, after performing a correlation analysis, that similarity metrics (including the novelty score used in NS) have a significantly positive correction with the t-wise coverage, which well explains the rationale of similarity testing. Moreover, we give detailed discussions on the suitability of NS for similarity-based SPL testing since NS perfectly matches the goal, i.e., searching for a set of diverse products. The suitability of NS and its superiority over some state-of-the-art approaches are demonstrated through comprehensive experimental studies performed on 31 feature models, either realistic or artificially generated, considering $t = 2, \ldots, 6$. In particular, it is found that NS significantly outperforms the state-of-the-art genetic algorithm [4] in 77% of all the test scenarios. Finally, concerning the way of generating new configurations, our results suggest that the use of two SAT solvers should be encouraged (while the use of genetic operators can be optional) in the context of similarity-based SPL testing.

Our results show that NS is promising for similarity-based testing of SPLs. In the future, it is possible to integrate objectives functions [8], such as test suite cost, coefficient of connectivity-density, into the framework of NS to handle objective requirements. Currently, we focus on using NS to generate product samples, without explicitly considering prioritization. Developing NS-based techniques which take both aspects into consideration is one of our subsequent studies. Finally, we intend to investigate whether looking for novelty can bring abundant benefits in other topics related to SPL testing, such as mutation-based test case generation [87], and behavioural SPL testing [46].

Acknowledgments

The authors would like to thank the anonymous reviewers for providing valuable comments to improve this paper.

References

[1] P. Clements and L. Northrop, *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[2] D. Batory, "Feature models, grammars, and propositional formulas," in *Proceedings of the 9th International Conference Software Product Lines, SPLC 2005*, H. Obbink and K. Pohl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 7–20.

[3] D. Benavides, S. Segura, and A. Ruiz-Corts, "Automated analysis of feature models 20 years later: A literature review ," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[4] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 650–670, July 2014.

[5] P. Knauber, J. Bermejo, G. Böckle, J. C. S. d. P. Leite, F. v. d. Linden, L. Northrop, M. Stark, and D. M. Weiss, *Quantifying Product Line Benefits*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 155–163.

[6] D. M. Weiss, "The product line hall of fame," in *The 12th International Software Product Line Conference*, 2008, pp. 395–395.

[7] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake, "Effective product-line testing using similarity-based product prioritization," *Software & Systems Modeling*, vol. 18, pp. 499–521, 2019.

[8] R. M. Hierons, M. Li, X. Liu, J. A. Parejo, S. Segura, and X. Yao, "Many-objective test suite generation for software product lines," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 1, pp. 2:1–2:46, Jan. 2020.

[9] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake, "Similarity-based prioritization in software product-line testing," in *Proceedings of the 18th International Software Product Line Conference - Volume 1*, ser. SPLC'14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 197–206.

[10] M. Varshosaz, M. Al-Hajjaji, T. Thüm, T. Runge, M. R. Mousavi, and I. Schaefer, "A classification of product sampling for software product lines," in *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, ser. SPLC'18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–13.

[11] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, 2008.

[12] M. F. Johansen, Ø. Haugen, and F. Fleurey, "Properties of realistic feature models make combinatorial testing of product lines feasible," in *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 638C652.

[13] R. E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed, "A first systematic mapping study on combinatorial interaction testing for software product lines," in *IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2015, pp. 1–10.

[14] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[15] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ser. ISSTA'07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 129–139.

[16] S. Oster, F. Markert, and P. Ritter, "Automated incremental pairwise testing of software product lines," in *Proceedings of the International Software Product Line Conference (SPLC)*, J. Bosch and J. Lee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 196–210.

[17] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing," *Empirical Software Engineering*, vol. 16, no. 1, pp. 61–102, 2011.

[18] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233–235, 1979.

[19] M. F. Johansen, Ø. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *Proceedings of the 16th International Software Product Line Conference - Volume 1*, ser. SPLC'12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 46–55.

[20] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of acts: A case study," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 591–600.

[21] M. Al-Hajjaji, S. Krieter, T. Thüm, M. Lochau, and G. Saake, "IncLing: Efficient Product-Line Testing Using Incremental Pairwise Sampling," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, ser. GPCE 2016. New York, NY, USA: Association for Computing Machinery, 2016, pp. 144–155. [Online]. Available: https://doi.org/10.1145/2993236.2993253

[22] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary search-based test generation for software product line feature models," in *Proceedings of International Conference on Advanced Information Systems Engineering (CAiSE)*, J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 613–628.

[23] D. Marijan, A. Gotlieb, S. Sen, and A. Hervieu, "Practical pairwise testing for software product lines," in *Proceedings of the 17th International Software Product Line Conference*, ser. SPLC'13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 227–235. [Online]. Available: https://doi.org/10.1145/2491627.2491646

[24] A. Cmyrev and R. Reissing, "Efficient and effective testing of automotive software product lines," *Applied Science and Engineering Progress*, vol. 7, no. 2, pp. 53–57, 2014.

[25] T. Pett, T. Thüm, T. Runge, S. Krieter, M. Lochau, and I. Schaefer, "Product sampling for product lines: The scalability challenge," in *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, ser. SPLC'19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 78–83.

[26] A. Arcuri and L. Briand, "Formal analysis of the probability of interaction fault detection using random testing," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1088–1099, 2012.

[27] M. Grindal, J. Offutt, and S. Andler, "Combination testing strategies: A survey," *Software Testing, Verification, and Reliability*, vol. 15, pp. 167–199, 2005.

[28] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, and S. Apel, "A comparison of 10 sampling algorithms for configurable systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 643–654.

[29] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 461–470.

[30] C. Song, A. Porter, and J. S. Foster, "itree: Efficiently discovering high-coverage configurations using interaction trees," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 251–265, 2014.

[31] R. Schröter, S. Krieter, T. Thüm, F. Benduhn, and G. Saake, "Feature-model interfaces: The highway to compositional analyses of highly-configurable systems," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE 16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 667–678.

[32] M. F. Johansen, Ø. Haugen, F. Fleurey, E. Carlson, J. Endresen, and T. Wien, "A technique for agile and automatic interaction testing for product lines," in *International Conference on Testing Software and Systems*, B. Nielsen and C. Weise, Eds. Springer Berlin Heidelberg, 2012, pp. 39–54.

[33] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *IT Professional*, vol. 10, no. 3, pp. 19–23, 2008.

[34] J. Petke, S. Yoo, M. B. Cohen, and M. Harman, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, pp. 26–36.

[35] E. Reisner, C. Song, K.-K. Ma, J. S. Foster, and A. Porter, "Using symbolic evaluation to understand behavior in configurable software systems," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE'10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 445–454.

[36] E. G. Cartaxo, P. D. L. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011.

[37] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 1, pp. 6:1–6:42, Mar. 2013.

[38] H. Hemmati and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *2010 IEEE 21st*

International Symposium on Software Reliability Engineering, 2010, pp. 141–150.

[39] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–10.

[40] S. Fischer, R. E. Lopez-Herrejon, R. Ramler, and A. Egyed, "A preliminary empirical assessment of similarity for combinatorial iteraction testing of software product lines," in *2016 IEEE/ACM 9th International Workshop on Search-Based Software Testing (SBST)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2016, pp. 15–18.

[41] A. B. Sánchez, S. Segura, J. A. Parejo, and A. Ruiz-Cortés, "Variability testing in the wild: the drupal case study," *Software & Systems Modeling*, vol. 16, no. 1, pp. 173–194, Feb 2017.

[42] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Assessing software product line testing via model-based mutation: An application to similarity testing," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 188–197.

[43] M. Al-Hajjaji, S. Lity, R. Lachmann, T. Thüm, I. Schaefer, and G. Saake, "Delta-oriented product prioritization for similarity-based product-line testing," in *2017 IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design (VACE)*, 2017, pp. 34–40.

[44] M. Al-Hajjaji, M. Schulze, and U. Ryssel, "Similarity analysis of product-line variants," in *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, ser. SPLC'18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 226–235.

[45] X. Devroey, G. Perrouin, A. Legay, P.-Y. Schobbens, and P. Heymans, "Covering SPL Behaviour with Sampled Configurations: An Initial Assessment," in *Proceedings of the Ninth International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VaMoS'15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 59–66. [Online]. Available: https://doi.org/10.1145/2701319.2701325

[46] X. Devroey, G. Perrouin, A. Legay, P. Schobbens, and P. Heymans, "Search-Based Similarity-Driven Behavioural SPL Testing," in *Proceedings of the Tenth International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VaMoS'16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 89–96.

[47] S. Lity, M. Al-Hajjaji, T. Thüm, and I. Schaefer, "Optimizing product orders using graph algorithms for improving incremental product-line analysis," in *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VAMOS'17. New York, NY, USA: Association for Computing Machinery, 2017, p. 60C67. [Online]. Available: https://doi.org/10.1145/3023956.3023961

[48] R. Lachmann, S. Lity, M. Al-Hajjaji, F. Fürchtegott, and I. Schaefer, "Fine-grained test case prioritization for integration testing of delta-oriented software product lines," in *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*, ser. FOSD 2016. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3001867.3001868

[49] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI)*, 2008, pp. 329–336.

[50] J. Lehman and O. S. Kenneth, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary Computation*, vol. 19, no. 2, pp. 189–223, 2011.

[51] S. Doncieux, A. Laflaquière, and A. Coninx, "Novelty search: A theoretical perspective," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO'19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 99–106.

[52] S. Risi, C. Hughes, and K. O. Stanley, "Evolving plastic neural networks with novelty search," *Adaptive Behavior*, vol. 18, no. 6, pp. 470–491, 2010.

[53] J. R. Romero, A. Ramírez, and C. L. Simons, "Looking for novelty in SBSE problems," in *Proceedings of the Spanish Conference on Software Engineering and Databases (JISBD)*, ser. JISBD'19, 2019, pp. 1–4. [Online]. Available: http://hdl.handle.net/11705/JISBD/2019/028

[54] M. Boussaa, O. Barais, G. Sunyé, and B. Baudry, "A novelty search approach for automatic test data generation," in *Proceedings of the Eighth International Workshop on Search-Based Software Testing*, ser. SBST'15. IEEE Press, 2015, pp. 40–43.

[55] V. R. López-López, L. Trujillo, and P. Legrand, "Novelty Search for Software Improvement of a SLAM System," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO'18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1598–1605.

[56] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[57] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel, "Distance-based sampling of software configuration spaces," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE'19. IEEE Press, 2019, pp. 1084–1094. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00112

[58] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *The 37th International Conference on Software Engineering*, vol. 1, May 2015, pp. 517–528.

[59] Y. Xiang, Y. Zhou, Z. Zheng, and M. Li, "Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers," *ACM Transactions on Software Engineering and Methodology*, vol. 26, no. 4, pp. 14:1–14:46, Feb. 2018.

[60] Y. Xiang, X. Yang, Y. Zhou, Z. Zheng, M. Li, and H. Huang, "Going deeper with optimal software products selection using many-objective optimization and satisfiability solvers," *Empirical Software Engineering*, vol. 25, pp. 591–626, 2020.

[61] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "PLEDGE: A Product Line Editor and Test Generation Tool," in *Proceedings of the 17th International Software Product Line Conference Co-Located Workshops*, ser. SPLC'13 Workshops. New York, NY, USA: Association for Computing Machinery, 2013, pp. 126–129.

[62] D. L. Berre and A. Parrain, "The Sat4j library, release 2.2, system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010.

[63] A. Balint and U. Schöning, *Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break*. Berlin, Heidelberg: International Conference on Theory and Applications of Satisfiability Testing, 2012, pp. 16–29.

[64] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VaMoS '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. Article No.: 7, PP. 1–8.

[65] M. Mendonca, A. Wasowski, and K. Czarnecki, "SAT-based Analysis of Feature Models is Easy," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 231–240.

[66] G. Perrouin, S. Sen, J. K. B. Baudry, and Y. le Traon, "Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines," *IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 459–468, 2010.

[67] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 5, pp. 394–397, 1962.

[68] B. Selman, H. Levesque, and D. Mitchell, "A new method for solving hard satisfiability problems," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, ser. AAAI'92. AAAI Press, 1992, pp. 440–446.

[69] S. Cai, "Faster implementation for walksat," Queensland Research Lab, NICTA, Australia, Tech. Rep., June 2013.

[70] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, ser. AAAI '94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 337–343.

[71] P. Jaccard, "Etude comparative de la distribution florale dans une portion des alpes et des jura," *Bull. del la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.

[72] S. Doncieux, G. Paolo, A. Laflaquière, and A. Coninx, "Novelty search makes evolvability inevitable," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO'20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 85–93.

[73] M. Mendonca, M. Branco, and D. Cowan, "S.P.L.O.T.: Software Product Lines Online Tools," in *Proceedings of the 24th ACM SIG-*

PLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. New York, NY, USA: Association for Computing Machinery, 2009, pp. 761–62.

[74] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "FeatureIDE: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642312001128

[75] C. Dietrich, R. Tartler, W. Schröder-Preikshat, and D. Lohmann, "Understanding linux feature distribution," in *Proceedings of the 2012 Workshop on Modularity in Systems Software*, ser. MISS'12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 15–20.

[76] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE'11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 1–10. [Online]. Available: https://doi.org/10.1145/1985793.1985795

[77] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *Information and Software Technology*, vol. 53, no. 5, pp. 407 – 423, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584910002193

[78] J. Lee, S. Kang, and D. Lee, "A survey on software product line testing," in *Proceedings of the 16th International Software Product Line Conference - Volume 1*, ser. SPLC'12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 31–40. [Online]. Available: https://doi.org/10.1145/2362536.2362545

[79] I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. de Almeida, "On strategies for testing software product lines: A systematic literature review," *Information and Software Technology*, vol. 56, no. 10, pp. 1183 – 1199, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914000834

[80] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba, *Evolutionary Computation for Software Product Line Testing: An Overview and Open Challenges*. Cham: Springer International Publishing, 2016, pp. 59–87. [Online]. Available: https://doi.org/10.1007/978-3-319-25964-2_4

[81] B. S. Ahmed, K. Z. Zamli, W. Afzal, and M. Bures, "Constrained interaction testing: A systematic literature study," *IEEE Access*, vol. 5, pp. 25 706–25 730, 2017.

[82] F. Ferreira, J. a. P. Diniz, C. Silva, and E. Figueiredo, "Testing tools for configurable software systems: A review-based empirical study," in *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VAMOS 19. New York, NY, USA: Association for Computing Machinery, 2019.

[83] J. Lee, S. Kang, and P. Jung, "Test coverage criteria for software product line testing: Systematic literature review," *Information and Software Technology*, vol. 122, p. 106272, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584920300227

[84] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/pii/000437028090051X

[85] A. Hervieu, B. Baudry, and A. Gotlieb, "PACOGEN: Automatic Generation of Pairwise Test Configurations from Feature Models," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, 2011, pp. 120–129.

[86] Z. Xu, M. B. Cohen, W. Motycka, and G. Rothermel, "Continuous test suite augmentation in software product lines," in *Proceedings of the 17th International Software Product Line Conference*, ser. SPLC'13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 52–61. [Online]. Available: https://doi.org/10.1145/2491627.2491650

[87] C. Henard, M. Papadakis, and Y. Le Traon, "Mutation-based generation of software product line test configurations," in *Proceedings of the 6th International Symposium on Search-Based Software Engineering (SSBSE 2014)*, 2014, pp. 92–106.

[88] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Using feature model knowledge to speed up the generation of covering arrays," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VaMoS'13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 16:1–16:6. [Online]. Available: https://doi.org/10.1145/2430502.2430524

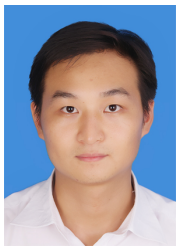
[89] C. H. P. Kim, D. S. Batory, and S. Khurshid, "Reducing combinatorics in testing product lines," in *Proceedings of the Tenth International Conference on Aspect-Oriented Software Development*, ser. AOSD'11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 57–68. [Online]. Available: https://doi.org/10.1145/1960275.1960284

[90] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Assessing software product line testing via model-based mutation: An application to similarity testing," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, March 2013, pp. 188–197.

[91] V. Tenev, S. Duszynski, and M. Becker, "Variant analysis: Set-based similarity visualization for cloned software systems," in *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B*, ser. SPLC'17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 22–27. [Online]. Available: https://doi.org/10.1145/3109729.3109753

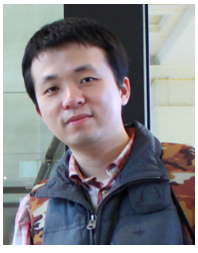
[92] T. Mende, R. Koschke, and F. Beckwermert, "An evaluation of code similarity identification for the grow-and-prune model," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 2, pp. 143–169, 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.402

[93] T. Yamamoto, M. Matsushita, T. Kamiya, and K. Inoue, "Measuring similarity of large software systems based on source code correspondence," in *Product Focused Software Process Improvement*, F. Bomarius and S. Komi-Sirviö, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 530–544.



Yi Xiang received the B.Sc. and M.Sc. degrees in mathematics from Guangzhou University, Guangzhou, China, in 2010 and 2013, respectively, and the Ph.D. degree in computer science from Sun Yat-sen University, Guangzhou, in 2018. He is currently a Post-Doctoral Fellow working with Prof. X. W. Yang within the School of Software Engineering, South China University of Technology, Guangzhou. His current research interests include many-objective optimization and search-based software engineering.

Han Huang received the B.Man. degree in Information Management and Information System, in 2002, and the Ph.D. degree in Computer Science from the South China University of Technology (SCUT), Guangzhou, in 2008. Currently, he is a professor at School of Software Engineering in SCUT. His research interests include evolutionary computation, swarm intelligence and their application. Dr. Huang is a senior member of CCF and member of IEEE.



Miqing Li is an assistant professor at School of Computer Science at the University of Birmingham. His research is principally on multi-objective optimisation, where he focuses on developing population-based randomised algorithms (mainly evolutionary algorithms) for both general challenging problems (e.g. many-objective optimisation, constrained optimisation, robust optimisation, expensive optimisation) and specific challenging problems (e.g. those in software engineering, system engineering, product disassembly, post-disaster response, neural architecture search, reinforcement learning for games).

Sizhe Li was born in Fujian, China in 2000. He is an undergraduate of the School of Software Engineering, South China University of Technology, Guangzhou, China.

Xiaowei Yang received the B.S. degree in theoretical and applied mechanics, the M.Sc. degree in computational mechanics, and the Ph.D. degree in solid mechanics from Jilin University, Changchun, China, in 1991, 1996, and 2000, respectively. He is currently a Full Time Professor with the School of Software Engineering, South China University of Technology, Guangzhou, China. His current research interests include designs and analyses of algorithms for large-scale pattern recognitions, imbalanced learning, semi-supervised learning, support vector machines, tensor learning, and evolutionary computation. He has published over 100 journals and refereed international conference articles, including the areas of structural reanalysis, interval analysis, soft computing, support vector machines, and tensor learning.