Looking for novelty in search-based software product line testing
Xiang, Yi; Huang, Han; Li, Miqing; Li, Sizhe; Yang, Xiaowei

DOI: 10.1109/TSE.2021.3057853
License: Other (please specify with Rights Statement)

Document Version
Peer reviewed version

Citation for published version (Harvard):
https://doi.org/10.1109/TSE.2021.3057853

Download date: 15. Sep. 2023
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 optimizing similarity metrics using search-based algorithms, a small subset of test cases (i.e., products) as dissimilar as possible, thus potentially making more \( t \)-wise combinations covered. Prior work has 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 correlation analyses to investigate how similarity metrics are correlated with the \( t \)-wise coverage. We find that similarity metrics generally have significantly positive correlations 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 well matches the goal of similarity-based SPL testing. We find that the novelty score used in NS has (much) stronger positive correlations with the \( t \)-wise coverage than previous approaches relying on a genetic algorithm (GA) with a similarity-based fitness function. Experimental results on 31 software product lines validate the superiority of NS over GA, as well as other state-of-the-art approaches, concerning both \( t \)-wise coverage and fault detection capacity. Finally, we investigate whether it is useful to combine two satisfiability solvers when generating new individuals in NS, and how the performance of NS is affected by its key parameters. In summary, looking for novelty provides a promising way of sampling diverse test cases for SPLs.

Index Terms—Software product line testing, product sampling, 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 common set of features, with a feature being some aspect of system functionalities [1], and a product being a subset 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 compactly represented 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, Boeing and Toshiba applying SPLs to develop software [6], [7].

Despite the benefits that SPLs bring, new challenges are raised with respect to how to ensure the reliability of an SPL. In this regard, testing SPLs becomes crucial to avoid fault propagation to the derived products [8]. Indeed, a defect in a single feature may exist in thousands or even millions of products [6]. 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 valid products, but it is rarely possible in practice due to a huge number of 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 [7], 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 [9]. The combinatorial interaction testing [10], among others, is a prominent approach that has been introduced to reduce the size of test suites while achieving a certain coverage of feature combinations [11], [12]. It is based on the observation that most faults are caused by interactions among a small number of features [13]. In particular, \( t \)-wise sampling requires to find a minimal subset of products that cover all possible interactions among \( t \) features by at least one product under test [14]. Since this is an NP-hard problem, several greedy or metaheuristic approaches have been proposed to perform \( t \)-wise sampling, such as Chvatal [11], [15], CASA [16], MoSo-PoLiTe [17], ICPL [18], ACTS [19] and IncLing [20]. A review of product sampling tools for SPLs can be found in a recent survey [9].

However, existing \( t \)-wise sampling/testing techniques face the scalability issue [21]. For large real-world SPLs, they often run out of memory, do not terminate, or take too much running time [4], [21], [22], [23], [24]. Although...
t-wise sampling 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. For example, according to [18], 480 products are required to achieve a full 2-wise coverage for the Linux kernel FM (2.6.28.6-icse11) with 6,888 features [25]. In fact, the maximum number of t-wise combinations for n features is \( C_n^t \cdot 2^t \) [13]. 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 [25]. Moreover, there is a practical need to deal with high interaction strengths \( t > 2 \) [26], [27], [28]. Petke et al. [27] showed that higher-strength test suites are able to find more faults, and Kuhn et al. [13] indicated that almost all defects can be found for the 6-wise coverage. Given the above, the use of t-wise sampling may be restricted in practice as it struggles with large FMs and high interaction strengths.

As an avenue to circumvent the scalability issue faced by t-wise sampling, similarity-based testing [29], [30], [31] was introduced to test SPLs. It leads to two problems being investigated: similarity-based sampling [4], [32], [33] and similarity-based prioritization [6], [8], [34], [35], [36]. The former selects, by using similarity metrics, a subset of products to test, while the latter chooses a good order in which products should be tested. In this article, we focus on similarity-based sampling, which is significantly different from the aforementioned t-wise sampling where all t-wise combinations must be covered. Specifically, the t-wise sampling uses a test-end criterion, i.e., t-wise coverage, to determine when to stop adding further products into a sample. When the sampling is finished, a full t-wise coverage must be obtained. In contrast, similarity-based sampling adopts similarity metrics (e.g., the similarity-based fitness defined by Henard et al. [4]) as criteria for selecting suitable next products during sampling. At any time in the sample selection process, diversity among already selected products should be increased so as to achieve a given goal, such as maximizing fault detection rate or structural coverage criteria. Hence, similarity-based sampling offers no guarantees on a full t-wise coverage, and this may have some potential impact on the effectiveness of the sampled products regarding fault detection capacity [33] (we will experimentally investigate this in Section 5.3). As a natural consequence, the number of test cases and the termination in t-wise sampling algorithms cannot be manually controlled. Instead, similarity-based sampling is flexible in specifying the number of desired test cases and the running time [4]. This is particularly beneficial in cases where the sample size and/or time for sampling is limited.

In fact, there have been a number of works exploiting similarity in the context of SPL testing, e.g., [4], [6], [8], [20], [34], [35], [36], [37], [38], [39], [40]. Experimental results in these works demonstrated that improvements on similarity metrics indeed lead to a decent, if not full, t-wise coverage. It seems to be natural and intuitive that similarity metrics are closely correlated with t-wise coverage. However, no in-depth investigation has been conducted so far to confirm the above intuition. In this paper, we intend to confirm this by performing correlation analyses to explore how similarity metrics are correlated with t-wise coverage in the context of SPL testing. Moreover, we apply, for the first time, the novelty search (NS) algorithm [41], [42], [43] to search-based similarity-driven sampling of SPLs. Unlike classical objective-oriented evolutionary algorithms, e.g., GA adopted in [4], NS abandons objectives and promotes evolution through the search for “novelty”, which implies being different from individuals discovered previously. This well matches the goal of similarity-based testing, i.e., sampling a set of products that are as diverse as possible. In fact, Romero et al. [44] have recently pointed out that NS seems to perform better than other related algorithms.

In summary, this paper provides the following contributions:

1) We prove that the novelty score used in NS is a generalization of the similarity-based fitness defined by Henard et al. [4]. The novelty score, a kind of similarity metrics, measures the uniqueness of an individual, and is defined as the average distance from this individual to its \( N_0 \)-nearest neighbors in a population [42]. We show that the similarity-based fitness [4] is equivalent to the novelty score when \( N_0 = N \) (a particular case), where \( N \) is the population size. Moreover, our correlation analyses show that both similarity metrics have significantly positive correlations with the t-wise coverage in most cases. The above finding is important as it forms theoretical foundations of similarity-based sampling (as a surrogate of the t-wise sampling).

2) We show that, compared with the similarity-based fitness used in GA [4], novelty scores adopted by NS have (much) stronger positive correlations with the t-wise coverage. 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, implemented by the randomized SAT4J solver [4], forms a state-of-the-art strategy for generating new products in the SPL field [7], [32], [47], [48], [49], [50]. In this paper, we empirically demonstrate that combining two types of satisfiability solvers (i.e., the randomized SAT4J\(^1\) + probSAT\(^2\) [52]) in a similar way as in [49], [50] can be more effective than the unpredictable way when generating new products on some large-scale real-world FMs, such as the 2.6.28.6-icse11 Linux kernel [25]. 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, followed by literature review provided in Section 3. Next, Section 4 details the proposed method, and Section 5 reports

1. The SAT4J [51] is a conflict-driven clause learning (CDCL) solver.
2. The probSAT is a stochastic local search (SLS) solver.
on the empirical study. In Section 6, we discuss threats to validity. Finally, Section 7 concludes the paper and outlines possible research lines for future studies.

2 Concepts and notations

In this section, we present and illustrate key concepts and notations used in this paper.

2.1 Feature model configurations as test cases

This paper focuses on SPL testing in which an FM [2] is used as input. An FM is a 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], [9], [53], 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, and $\mathcal{C} = \{c_1, \ldots, c_k\}$ is a set of $k$ constraints among these features. $C$ is satisfied if and only if all $c_i$ ($i = 1, \ldots, k$) are satisfied.

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 also be called a product in the SPL terminology, an individual (or a solution point) in the evolutionary computation 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 $t$ features of an SPL [18], [54]. 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 [12].

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

A valid $t$-set, $tset$, is covered by a configuration $CF$, if $tset \subseteq CF$. Note that invalid $t$-sets need not to be covered [6].

Definition 6. The $t$-wise coverage of a test suite $TS = [CF_1, \ldots, CF_N]$ is defined as the ratio $\frac{|\bigcup_{i=1}^{N} VT_{CF_i}|}{|VT_{FM}|}$, where $VT_{FM}$ is the set of all the valid $t$-sets of the given FM; $VT_{CF_i}$ 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.

Definition 7. A $t$-wise covering array is a test suite, by which all valid $t$-sets are covered [14], [55].

2.3 An illustrative example

Fig. 1 depicts the FM of a raster graphics editor (RGE) product line [4]. This FM is adopted as an illustrative example to explain some key concepts described above. According to [56], each FM can be translated to a Boolean formula in conjunctive normal form (CNF). In this form, an FM is expressed as a conjunction of clauses, each of which is a disjunction of several literals. For example, the FM in Fig. 1 can be represented in the following CNF formula:

$$\begin{align*}
(\neg f_1 \lor \neg f_2) & \land (\neg f_3 \lor f_4) \\
& \land (\neg f_5 \lor f_6) \\
& \land (\neg f_7 \lor f_8) \\
& \land (\neg f_9) \\
& \land (\neg f_1 \lor \neg f_2) \\
& \land (\neg f_3 \lor f_4) \\
& \land (\neg f_5 \lor f_6) \\
& \land (\neg f_7 \lor f_8) \\
& \land (\neg f_9) \\
& \land (\neg f_1 \lor f_2) \\
& \land (\neg f_3 \lor f_4) \\
& \land (\neg f_5 \lor f_6) \\
& \land (\neg f_7 \lor f_8) \\
& \land (\neg f_9) \\
& \land (\neg f_1 \lor \neg f_2) \\
& \land (\neg f_3 \lor f_4) \\
& \land (\neg f_5 \lor f_6) \\
& \land (\neg f_7 \lor f_8) \\
& \land (\neg f_9) \neg f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9
\end{align*}$$

Clearly, each constraint in $C$ is a clause in the corresponding CNF formula. In the absence of constraints, $2^9 = 512$ configurations can be established. Taking into account constraints, however, only four of them are valid, and they are the following:

- $CF_1 = \{+f_1, +f_2, +f_3, +f_4, +f_5, -f_6, +f_7, -f_8, +f_9\}$
- $CF_2 = \{+f_1, +f_2, +f_3, +f_4, +f_5, +f_6, -f_7, -f_8, -f_9\}$
- $CF_3 = \{+f_1, +f_2, +f_3, +f_4, +f_5, +f_6, +f_7, -f_8, +f_9\}$
- $CF_4 = \{+f_1, +f_2, +f_3, -f_4, +f_5, +f_6, -f_7, +f_8, -f_9\}$

![Figure 1. The feature model, which is taken from [4], represents a raster graphics editor software product line.](image_url)
the combinations using SAT solvers and remove those that are invalid. For example, \{f_1, f_3, \neg f_6\} is a valid 3-set, while \{-f_1, f_3\} is not a valid 2-set because it violates \(f_1 \lor \neg f_3\). For this FM, it encompasses 73 valid 2-sets, 204 valid 3-sets, etc. The 2-wise coverage of the test suite \([CF_1, CF_4]\) is 66/73 \(\approx 90.4\%\), and the 3-wise coverage is 164/204 \(\approx 80.4\%\). For this model, the 2-wise covering array is \([CF_1, CF_2, CF_3, CF_4]\), which covers all valid 2-sets. It must be noted that the size of a covering array is not necessarily equal to the number of all valid products. This illustrative example is special as it encompasses only four valid products.

Finally, let us illustrate the differences between t-wise sampling and similarity-based sampling. Taking \(t = 2\) as an example, the generated test suite by 2-wise sampling is \([CF_1, CF_2, CF_3, CF_4]\), for which the 2-wise coverage is 100\%. Suppose that two test cases should be generated, the similarity-based sampling may return the test suite \([CF_1, CF_4]\), which covers 90.4\% of the 2-sets. Even though 100\% coverage is not obtained, this coverage is reasonably high considering other test suites, e.g., \([CF_1, CF_3]\) with a coverage of only 60.2\%.

3 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 [12], [57], [58], [59], [60], [61]. In particular, many t-wise sampling approaches have been proposed to generate minimal covering arrays. Broadly, they can be categorized as greedy algorithms (e.g., Chvatal [11], [15], ICPL [18], InCling [20]), metaheuristic algorithms (e.g., CASA [16], Ensan’s GA [62]), constraint programming (e.g., [17], [63], [64]). According to [24], however, most of the 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. As an alternative to t-wise sampling, the similarity-based sampling can bypass the scalability issues faced by t-wise sampling techniques. Moreover, it does not require more domain knowledge than the feature model, i.e., knowledge on valid combinations of features [6]. In the following part, we give an overview on several prominent similarity-based SPL testing techniques.

Henard et al. [4] applied similarity testing to SPLs in order to sample and prioritize products for especially large SPLs. The key idea is to mimic t-wise coverage by maximizing diversity among products. In their work, a genetic algorithm (GA) is designed to evolve a population of products by optimizing the similarity-based fitness function. More specifically, the algorithm starts with an initial population. Then they are prioritized using any prioritization method, and evaluated using the fitness function. The next step consists of attempting to replace the worst configuration by a new configuration. This replacement is permitted if and only if the similarity-based fitness increases. The above steps are repeated until the termination condition is satisfied. In the algorithm, SAT4J [51] is utilized to generate configurations during the search. However, Henard et al. [4] found that the internal order used by this solver to parse the logical clauses and literals leads to the same configuration in a deterministic way. To improve the randomness, and thus to get configurations in an unpredictable way, they randomized how the solver parses logical clauses and literals, and how it assigns values to variables. The resulting solver is called the randomized SAT4J in this paper. The unpredictable strategy implemented by this randomized solver serves as a comparison basis in [4], and also in our work due to its good scalability and high efficiency.

Results showed that Henard’s GA [4] is promising for large SPLs and high interaction strengths, forming a scalable and flexible alternative to the t-wise testing. This approach is supported by a test generation tool, called PLEDGE [32]. As an application of the similarity testing, Henard et al. [65] evaluated the capability of test suites to kill mutants of feature models, i.e., erroneous feature models derived from an original one. Experimental results demonstrated that dissimilar tests suites kill (or detect) more mutants than similar ones, thus validating the effectiveness of similarity-based SPL testing. Later, Fischer et al. [33] showed that Henard’s GA is useful to detect interaction faults in the Drupal [66] case study in which real fault data are used. They reported that this algorithm is useful to detect interaction faults in Drupal, being also able to yield a t-wise coverage comparable with CASA [16]. These findings further confirm that similarity is a suitable surrogate metric for the t-wise coverage.

Al-Hajjaji et al. systematically studied similarity-based prioritization in SPL testing in four related papers [6], [8], [34], [35]. 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 approach proposed by Al-Hajjaji et al. [6], [8] follows an incremental way to select configurations. More specifically, the product with the maximum number of selected features (i.e., all-yes-config [67]) is chosen at first. Then, this approach selects one by one the next product that has the maximum dissimilarity with the already selected products. The process continues until all products are prioritized. It should be noted that similarity between a product and those already chosen is measured by the minimum distance, rather than the sum of distances as in [4]. According to Al-Hajjaji et al. [6], [8], the sum of distances as a selection criterion may be misleading in some cases. They evaluated the effectiveness of similarity-based prioritization on SPLs with both real and artificial faults. However, one of the disadvantages of this incremental approach is that its time complexity is high, being \(O(N^3)\), where \(N\) is the size of a test suite.

Al-Hajjaji et al. [34] later extended the above prioritization approach by considering 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 [35], 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 simi-
larity). 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.

Devröy et al. [38] assessed the SPL behavioural coverage of configurations sampled by two structural testing approaches, i.e., t-wise testing and similarity-based 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 test suites, tools ICPL (t-wise) [18] and PLEDGE (similarity-based) [32] were used. In a subsequent study [39], 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. [36] 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], [6], [8], [34], [38] 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. [40] 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 [68]. There have been several approaches exploiting similarity to the source code level, e.g., [69], [70], [71]. As pointed out by Al-Hajjaji et al. [35], 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.

In addition to similarity-based SPL sampling, uniform sampling, in which every valid configuration has an equal chance of being chosen [72], provides an alternative to sampling products for SPLs. Smarch [72], [73] is the latest uniform sampling algorithm for SPLs. It creates a one-to-one mapping that converts a uniformly sampled integer into a unique configuration in the whole configuration space of the propositional formula derived from an FM. Although Smarch gives strong guarantees of uniformity, evidences in [72] show that it can be expensive to run, especially on large-scale SPLs. Moreover, Oh et al. [73] evaluated the t-wise coverage of samples produced by Smarch. They found that uniform sampling alone is not enough to produce 100% t-wise coverage. In this paper, Smarch is included in our performance evaluations as we intend to investigate how similarity-based sampling performs, in terms of covering t-wise combinations, when compared with uniform sampling.

Finally, it is worth mentioning that SPL testing goes beyond the product sampling/prioritization. In this paper, we do not intend to cover full SPL testing process, but focus particularly on product sampling for SPLs, an open research issue [4], [59], [60].

4 Novelty search for SPL testing

The NS algorithm for sampling products in SPL testing, as outlined in Algorithm 1, aims at looking for a specified amount of diverse configurations (maintained in an archive \( \mathcal{A} \)), 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 NS.

- The archive size (or test suite size, \( N \)) specifies the number of configurations to be returned. The setting of this parameter depends largely on demands of software testers.
- The execution time (\( \text{max._t} \)) 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 (if necessary), with the probability \( P_r \), using the probSAT solver [52]. Proper values of this parameter will be experimentally investigated later in Section 5.5.1.
- The neighbor size (\( \mathcal{N}_b \)), an important parameter in NS, specifies the number of neighbors to be considered when evaluating the “novelty” of a configuration. Empirical studies will be conducted in Section 5.5.2 to tune this parameter.

Notice that the NS algorithm allows software testers to flexibly specify 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.

4.1 Initialization of the archive

According to Line 1 of Algorithm 1, \( \mathcal{A} \) is initialized with \( N \) valid configurations produced in an unpredictable way. As mentioned in Section 3, the unpredictable way [4] relies on the randomized SAT4J\(^4\) solver. To implement this 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\(^5\), and therefore significantly improves the diversity of the generated configurations [4], [48]. Being simple and

4. SAT4J library is available at \( \text{https://www.sat4j.org} \).
5. The randomized order within the SAT4J solver is defined by ((Solver)reasoner(SATsolverGetSolver())).setOrder(new RandomWalkDecorator(new VarOrderHeap(new RandomLiteralSelectionStrategy))).1). More details can be found at \( \text{https://henard.net/research/SPL/TSF_2014/} \). To further improve diversity, PositiveLiteralSelectionStrategy and NegativeLiteralSelectionStrategy can be complementally used, please refer to [48] for details.
effective, the above unpredictable way forms a state-of-the-art strategy to sample configurations from the space of all the valid products [47]. In fact, this strategy has been widely used to generate configurations in the context of both optimal software product selection [48], [49], [50] and software product line testing [7], [32]. In this paper, we also adopt the 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.

4.2 Generation of new configurations

In the iteration process (Lines 3-17 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 \( c \), 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 [52] with the probability \( P_r \) (see Line 6), or replaced by an unpredictable configuration obtained from the randomized SAT4J solver (see Line 8).

The procedure of probSAT [52], an SLS-based SAT solver\(^6\) [75], is given in Algorithm 2. Essentially, it iteratively flips a selected variable (by changing its value from 0 to 1, or vice versa) until the number of flips reaches the maximum value, \( \text{max} \text{.flips} \). For an invalid configuration \( c \), it violates at least one clause. As shown in Line 3 of Algorithm 2, we randomly pick one of these falsified clauses, denoted by \( \text{cls} \). Next, the procedure works out the break value of each variable \( v \) in \( \text{cls} \) (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 [76]. 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 [76]. For details, please refer to the original study.

Subsequently, as shown in Line 6, break \( (v) \) is transformed by the following polynomial function\(^7\).

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

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

\[\text{Algorithm 2 Procedure of the probSAT solver}\]
\[\text{Input:} \text{an invalid configuration} \ c, \ \text{maximum number of flips} \ (\text{max} \text{.flips})\]
\[\text{Output:} \text{the repaired} \ c\]
\begin{align*}
1 & \text{n} \leftarrow 0 / / \text{the number of flips} \\
2 & \text{while} \ n \leq \text{max} \text{.flips} \text{ do} \\
3 & \text{Randomly pick a falsified clause} \ \text{cls} \text{ with respect to the configuration} \ c \\
4 & \text{for each variable} \ v \text{ in} \ \text{cls} \text{ do} \\
5 & \text{Compute the break value of} \ v, \ i.e., \ \text{break} (v) \\
6 & \text{Transform} \ \text{break} (v) \text{ according to Eq. (1)} \\
7 & \text{end for} \\
8 & \text{var} \leftarrow \text{the variable selected based on the probability} \\
9 & \frac{1}{\sum \text{break}(z)} \\
10 & \text{Flip the variable} \ \text{var} \text{ in} \ c \\
11 & \text{n} \leftarrow \text{n} + 1 \\
12 & \text{end while} \\
13 & \text{return} \ c
\end{align*}

According to Line 8 of Algorithm 2, the variable \( \text{var} \) to be flipped is chosen based on the probability \( \frac{1}{\sum \text{break}(z)} \). This calculation of probability ensures that variables with larger \( f(v) \) 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 must be mentioned that repairing SPL configurations using probSAT has already been explored in our previous work [50] in the context of multi-objective software product selection from SPLs. It has been shown that probSAT is more effective than WalkSAT [77], 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 this study.

\[\text{6. In addition to SLS-style solvers, another mainstream of high-performance algorithms for satisfiability solving is CDCL solvers [74], such as SAT4J. Different from CDCL-style solvers, which need to methodically traverse the whole search space, SLS-style solvers are typically greedy algorithms aiming at quickly satisfying clauses as many as possible. However, they have no guarantee on finding a satisfying solution for each solver call.}\]

\[\text{7. According to [52], break} (v) \text{ can be transformed by other types of functions. The polynomial function is chosen in this study primarily because of its simplicity and effectiveness [52].}\]
the context of SPL testing. We make the first step in this regard. Most importantly, as will be demonstrated in Section 4.2, the introduction 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) [25] which is the largest FM widely studied before.

In the detailed implementation, max_flips is set to 4,000, following the common practice in [49], [50]. The parameter $c_0$ 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.

Finally, in case that the random configuration $c$ 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 4.1, the randomized solver enables an exploration of the valid search space in an unpredictable way, being capable of improving diversity of configurations.

4.3 Evaluation of novelty

According to Line 12 in Algorithm 1, the generated configuration $c$ (after repair or replacement) and all the archived members are jointly evaluated for novelty. For any $x \in \mathcal{A} \cup \{c\}$, the novelty score of $x$ is given by

$$\rho(x) = \frac{1}{N_b} \sum_{i=1}^{N_b} d(x, \mu_i),$$

where $d(x, \mu_i)$ is the distance between $x$ and its $i$-th nearest neighbor, $\mu_i$. According to (2), the novelty score is defined as the average distance to the $N_b$-nearest neighbors of a configuration. In fact, it 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 [42]. This is reason why we consider the set $\mathcal{A} \cup \{c\}$, instead of $\mathcal{A}$, when evaluating the novelty of an individual.

Since both $x$ and $\mu_i$ are sets according to Definition 2, any set-based distance metric can be applied here. We consider the following distance metrics given their popularity in this field.

1) Jaccard distance:

$$d_{\text{Jaccard}}(x, \mu_i) = 1 - \frac{|x \cap \mu_i|}{|x \cup \mu_i|},$$

where $|\cdot|$ denotes the cardinality of a set. This distance metric was adopted in [4] to measure the degree of similarity between two configurations.

2) Hamming distance:

$$d_{\text{Hamming}}(x, \mu_i) = 1 - \frac{|x \cap \mu_i|}{|\mathcal{F}|},$$

where $|\mathcal{F}|$ is the number of all features. This distance metric was adopted by Al-Hajji et al. [6] to prioritize products.

3) Dice distance [39]:

$$d_{\text{Dice}}(x, \mu_i) = 1 - \frac{2|x \cap \mu_i|}{|x \cup \mu_i| + |x \cap \mu_i|}.$$

4) Anti-dice distance [39]:

$$d_{\text{Anti-dice}}(x, \mu_i) = 1 - \frac{|x \cap \mu_i|}{2|x \cup \mu_i| - |x \cap \mu_i|}.$$

Values for all the above distance metrics vary 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. We illustrate how these distances are calculated using the running example presented in Section 2.3. Considering $CF_1$ and $CF_{39}$,

$$d_{\text{Jaccard}}(CF_1, CF_3) = 1 - \frac{8}{10} = 0.2.$$

Similarly,

$$d_{\text{Hamming}}(CF_1, CF_3) = 1 - \frac{8}{9} \approx 0.11,$$

$$d_{\text{Dice}}(CF_1, CF_3) = 1 - \frac{2 \times 8}{10 + 8} \approx 0.11,$$

$$d_{\text{Anti-dice}}(CF_1, CF_3) = 1 - \frac{8}{2 + 10 - 8} \approx 0.33.$$

Notice that influence of these distance metrics has already been explored by Devroy et al. [39] in the context of similarity-based test generation for behavioural SPL models. In this work, we also examine these metrics given their high popularity. In Section A-II of the appendix, we experimentally compared these distance metrics. It is found that Anti-dice distance obtains the best overall performance, and is particularly effective on real-world FMs. Therefore, this distance metric is recommended to be used in the context of SPL testing.

At the end of this section, we generalize the calculation of novelty score from a single configuration to a test suite using Eq. (3). It maps a test suite to a positive real value, which is the sum of novelty scores over all members in a test suite.

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

where $TS$ is a test suite with $N$ configurations, i.e., $TS = [CF_1, \ldots, CF_N]$, and $\rho(CF_i)$ is the novelty score of the single configuration, $CF_i$, as given in Eq. (2). In Section 5.1, we will investigate correlations between a test suite’s novelty score and its t-wise coverage.

4.4 Reward novel individuals

As shown in Line 14 of Algorithm 1, the worst member in $\mathcal{A}$, $c_{\text{worst}}$, is identified by comparing novelty scores. After that, $c_{\text{worst}}$ will be replaced by the newly generated $c$ in case that $\rho(c) > \rho(c_{\text{worst}})$. 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 is important to note that NS generally follows a similar framework as in typical search-based algorithms (e.g., evolutionary algorithms) [42], [43]. It does not impose strong constraints on how new individuals are produced. Crossover and mutation operators widely used in evolutionary algorithms can be also applied to NS. The essential difference is that in NS there is no objective function associated with the problem, that is, we do not need to optimize (minimize or maximize) a specific fitness function; instead, the goal is to search for new (or novel) individuals that have the maximum difference from the solutions we already found. In fact, this is precisely what we pursue in SPL testing.

4.5 Why Novelty Search?

As discussed in [44], NS is well suitable for SBSE problems. Going one step further, we demonstrate in this paper that NS is an ideal tool for similarity-based testing of SPLs because of the following facts.

First, in SPL testing, a set of test cases is 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 SPLs.

Second, 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.

Third, previous works [4], [8] on SPL testing ambiguously adopted 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 it is more straightforward, and more importantly, it has some good theoretical properties, e.g., behaving like a uniform random search process in the behavior space [43] and creating a pressure for high evolvability even in bounded behavior spaces [78]. These properties are very useful in guiding the search towards diverse configurations during the sampling of SPLs. In fact, as will be shown in Section 5.1.1, the novelty score used in NS generalizes the similarity-based fitness function [4] by making the number of nearest neighbors a parameter of the score.

Finally, experiments presented in the next section show that NS can indeed obtain promising results in testing SPLs with respect to both t-wise coverage and fault detection rate.

5 Empirical study

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

RQ1: What is the correlation between similarity metrics and t-wise coverage?

RQ2: How does NS perform, in terms of t-wise coverage, when compared with state-of-the-art algorithms for similarity-based SPL testing?

RQ3: How does NS perform, in terms of fault detection rate, when compared with state-of-the-art algorithms for similarity-based SPL testing?

RQ4: Is it useful to combine two types of SAT solvers within NS when generating new configurations?

RQ5: How is the performance of NS, regarding t-wise coverage, affected by its key parameters?

The first research question is a foundational one in similarity-based testing of SPLs. Indeed, if similarity metrics have no correlations with the t-wise coverage, then it is meaningless to optimize similarity metrics so as to achieve a decent t-wise coverage. To address RQ1, correlation analyses are performed to investigate internal correlations 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 NS concerning t-wise coverage in comparison with some state-of-the-art approaches. We expect NS to provide t-wise coverage better than or close to these approaches. As mentioned in Section 1, the effectiveness of similarity-based sampling, in terms of fault detection capacity, can be potentially affected. The third research question aims at investigating these impacts and comparing NS against several state-of-the-art algorithms. The fourth research question explores whether it is useful to combine two types of SAT solvers when generating new configurations. Finally, the fifth research question seeks to provide useful guidelines for tuning 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. 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 [79] or the LVAT repository [25]. In this work, we add three moderate FMs, i.e., E-shop from SPLOT; toybox and aXTLS from LVAT, and two large FMs, i.e., Automotive01 and Automotive02 [21]. The two large FMs are closed-source product lines from automotive industry, and well suited as subjects for examining the scalability of t-wise testing algorithms [21]. Regarding the FMs, 20 of them are real, while 11 are artificially generated.

8. SPLOT is available at http://www.splot-research.org/
9. LVAT is available at http://code.google.com/p/linux-variability-analysis-tools
10. Both Automotive01 and Automotive02 (in V3 version) are integrated in FeatureIDE [80] (https://featureide.github.io/). We used the export functionality of FeatureIDE to generate these models’ propositional formulas as DIMACS files.
by the SPLIT FM generator [79], with the prefix being “SPLIT-Generated-FM” in their names.

For each FM, Table 1 presents its name, the number of features, the number of CNF constraints, the number of core and dead features, the number of redundant constraints (as well as the ratio to the number of all constraints) and the number of valid 2-sets. Note that a core feature is the feature which must present in every valid configuration. In contrast, a dead feature cannot be included in any product. Here the number of core and dead features is computed by PLEDGE [32]. Using core and dead features, constraints can be simplified by using the procedure as described in [49]. Then, the number of redundant constraints (as well as the ratio) can be easily worked out after simplification.

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. How to compute \( t \)-sets in this case is already illustrated in Section 2.3 using the presented running example. For the large FMs, it is very time-consuming to obtain all the valid \( t \)-sets even for \( t = 2 \) [4]. To compute \( t \)-wise coverage in this case, we instead randomly generate a large set of valid \( t \)-sets, which can be seen as an approximation of the whole space of all the valid \( t \)-sets. The steps of sampling random \( t \)-sets are as follows [6]. First, we randomly pick \( t \) features from the feature set \( \mathcal{F} \). Then, we decide for each picked feature randomly whether it is selected or deselected. Next, we check whether this feature combination is valid or not using SAT solvers. The above steps are repeated until a desired number of valid \( t \)-sets are generated. Since the number of all valid \( t \)-sets increases with \( t \), we let the number of randomly generated \( t \)-sets be 25,000. Notice that, for some very small FMs (e.g., CounterStrikeSimpleFM) and relatively low \( t \) (e.g., \( t \leq 4 \)), the number of all valid \( t \)-sets is lower than the above number. In this case, we enumerate all valid \( t \)-sets as done for \( t = 2 \). According to our setting, more \( t \)-sets will be generated as \( t \) increases. For example, we generate 75,000 for \( t = 3 \), and 100,000 valid \( t \)-sets for \( t = 4 \), etc. To take randomness into account, 10 sampling sets are generated for each FM and each value of \( t \), and the \( t \)-wise coverage of a test suite is averaged over these sampling sets.

### 5.1 Correlation analysis between novelty score and \( t \)-wise coverage (RQ1)

In this section, we perform correlation analyses to investigate correlations between novelty scores and \( t \)-wise coverage. We start by showing how the novelty score of a test suite generalizes the similarity-based fitness defined by Henard et al. [4]. Next, we present experimental setup and results of the correlation analysis. Finally, answers to RQ1 are given.

#### 5.1.1 Novelty score generalizes the similarity-based fitness

Using Eqs. (2) and (3), the novelty score for a test suite, \( \rho(TS) \), can be rewritten as

\[
\rho(TS) = \frac{1}{N_b} \sum_{i=1}^{N_b} \sum_{j=1}^{N_b} d(CF_i, CF_{ij}),
\]

where \( CF_{ij} \) is the \( j \)-th nearest neighbor of \( CF_i \). Clearly, for a fixed \( TS \), values of \( \rho(TS) \) are highly related to the setting of \( N_b \). For instance, in case of \( N_b = N \), \( \rho(TS) \) is equivalent.
The proof is straightforward. Since $N_b = N$, $\rho(TS)$ in (4) can be reformulated as

$$\rho(TS) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=i}^{N} d(CF_i, CF_j). \tag{6}$$

Because $d(CF_i, CF_j) = d(CF_j, CF_i)$, $\rho(TS)$ can be further written as

$$\rho(TS) = \frac{2}{N} \sum_{i=1}^{N} \sum_{j=i+1}^{N} d(CF_i, CF_j) = \frac{2}{N} f(TS). \tag{7}$$

to the following similarity-based fitness function suggested by Henard et al. [4].

Figure 2. Correlation analysis results, shown in boxplots. The dotted line in each plot connects the average of the values.
Obviously, for the same test suite, \( \rho(TS) \) and \( f(TS) \) give the same value, except for the fixed factor \( 2/N \). Therefore, by introducing the parameter \( N_b \), the novelty score generalizes the similarity-based fitness function [4]. Since the former is more generic, its use is advisable.

Intuitively, according to [4], the higher the similarity-based fitness (or broadly the novelty score) of a given TS, the larger the distance among configurations, leading to potentially higher \( t \)-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 correlation analyses.

### 5.1.2 Experimental setup

To perform correlation analysis experiments, we generate 1000 test suites for each FM, and each of which contains \( N \) configurations. Following the practice in [4], we consider three typical values for \( N \), i.e., 50, 100 and 500. This allows us observing the trend as \( N \) increases. It should be noted that these configurations are generated using the unpredictable strategy suggested by Henard et al. [4]. After each test suite is generated, we calculate its \( t \)-wise coverage \( (t = 2, \ldots, 6) \) and novelty scores with varying values for \( N_b \). Precisely, for each \( N \), we consider six values for \( N_b \), i.e., 2, 15, \( N/4 \), \( N/2 \), 3\( N/4 \) and \( N \). Notice that 2 is the minimum possible value for \( N_b \). When \( N_b = 2 \), the novelty score of a configuration is evaluated based on its two closest neighbors (including itself). The value 15 is considered because it has been widely adopted in NS-related literature [81]. As mentioned before, \( N_b = N \) leads to a special case in which the novelty score is equivalent to the similarity-based fitness. In addition, we examine the other three values, i.e., \( N/4 \), \( N/2 \) and 3\( N/4 \). This way, for each \( FM \) and each \( t \), we can perform a correlation analysis to observe how and, to what extent, the \( t \)-wise coverage is correlated with the novelty score under each setting of \( N_b \).

### 5.1.3 Results of correlation analysis

The correlation analysis returns Pearson’s correlation coefficient \( r \) and \( p \)-values for testing the hypothesis that there is no correlation between two random variables (null hypothesis). The Pearson’s \( r \) 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 (default is 0.05) indicate that the corresponding correlation is considered significant. The correlation analysis results are available online\(^{11}\). For each \( FM \) and each value of \( t \), we provide the average \( t \)-wise coverage (over 1000 test suites), Pearson’s \( r \) values and \( p \)-values for all settings of \( N_b \). To make analysis of these data easy, we present them in the form of boxplots. For each pair of \( (N, t) \), as shown in Fig. 2, a boxplot is used to report correlation analysis results. Specifically, the \( x \)-axis denotes different settings for \( N_b \), and \( y \)-axis presents Pearson’s \( r \) values for all the 31 FMs. The number on top of the boxes indicates the number of statistically significant \( r \) values (\( p < 0.05 \)).

As can be seen from Fig. 2, Pearson’s \( r \) values are obviously larger than 0 in the vast majority of the cases. This clearly suggests that novelty scores have positive correlations with the \( t \)-wise coverage. For a better visualization of their correlations, we plot in Fig. 3 (in the form of scatter plots) the novelty score (\( x \)-axis) and \( t \)-wise coverage (\( y \)-axis) for all the randomly generated test suites on two representative FMs (CounterStrikeSimpleFM and aXTLS). Note that, in the calculation of the novelty score, we choose \( N_b = N/2 \) as an example, where \( N = 100 \). As observed, the \( t \)-wise coverage generally increases along with the novelty score, and this is particularly obvious for large \( t \) values.

We are also aware that \( r \) values are slightly smaller than 0 in some cases. For example, negative \( r \) values are observed on some small- and moderate-scale FMs for \( t = 2, 3, 4 \) when \( N = 500 \). However, the corresponding \( p \)-values indicate that almost all of the negative correlations are statistically insignificant (\( p > 0.05 \))\(^{12}\). On the contrary, positive correlations are mostly statistically significant (\( p < 0.05 \)). In fact, the number of significantly positive correlations is 924, 905 and 749 for \( N = 50 \), \( N = 100 \) and \( N = 500 \), respectively. Note that, for each \( N \) value, there are 31 FMs, five values of \( t \) and six settings for \( N_b \), leading to \( 31 \times 5 \times 6 = 930 \) cases in total. Considering three values for \( N \), the total number of cases is up to 2790. We can then work out the percentage of significantly positive correlations as follows: \( (924 + 905 + 749)/2790 = 92.401\% \). Clearly, novelty scores have significantly positive correlations with the \( t \)-wise coverage in the majority of cases.

Moreover, some interesting trends are observed as \( N \), \( t \) and \( N_b \) increase. In the following part, we will describe these trends and give possible explanations for them. For a fixed \( t \), as can be seen from Fig. 2, the number of statistically significant \( r \) values tends to decrease as \( N \) increases. This is particularly true for small \( t \) values, e.g., \( t = 2 \) and \( t = 3 \). More precisely, Table 2 summarizes the percentages of significantly positive correlations for different \( N \) values. When \( N = 50 \), as shown in Table 2, significantly positive correlations for small-, moderate- and large-scale FMs account for 38.065\%, 41.936\% and 19.355\% of the total 930

---

\(^{11}\) http://doi.org/10.5281/zenodo.4243159

\(^{12}\) In the provided online material, we find only five cases in which significantly negative correlations are observed. See the records: \((N = 100, \text{FM} = \text{SmartHome2.2}, t = 2, N_b = 2)\) and \((N = 500, \text{FM} = \text{ElectronicDrum}, t = 2, N_b = \{N/4, N/2, 3N/4, N\}\).
cases, respectively. And the sum reaches up to 99.356%. For $N = 100$ and $N = 500$, the cumulative percentages are 97.313% and 80.538%, respectively. Clearly, a descending trend is observed as $N$ increases. Furthermore, it can be seen from Table 2 that the decline on small-scale FMs is more obvious than on moderate- and large-scale ones. The above phenomenon can be explained as follows. As $N$ increases, a test suite can potentially cover (much) more $t$-sets. When $N$ is large enough, any randomly generated test suite may be able to achieve (nearly) full $t$-wise coverage, especially on small-scale FMs for small $t$ values. In this case, correlations between the $t$-wise coverage and novelty scores can be weak as the same (full) coverage is obtained no matter what values of the novelty score are. Naturally, the number of statistically insignificant correlations potentially increases as $N$ rises. The above explanation conforms to the results presented in Table A-1 in the appendix, where we list all the records in which statistically insignificant $r$ values ($p > 0.05$) are observed for all settings of $N_b$. As seen, no such record exists for $N = 50$, and only one for $N = 100$. The number of such records, however, rises up to 20 for $N = 500$. For all these records, being in line with the above explanation, (nearly) full $t$-wise coverage is obtained, and they are mostly related to small-scale FMs and small $t$ values.

<table>
<thead>
<tr>
<th>N = 50</th>
<th>N = 100</th>
<th>N = 500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small FMs</td>
<td>38.065%</td>
<td>36.022%</td>
</tr>
<tr>
<td>Moderate FMs</td>
<td>41.936%</td>
<td>41.936%</td>
</tr>
<tr>
<td>Large FMs</td>
<td>19.355%</td>
<td>19.355%</td>
</tr>
</tbody>
</table>

Table 2: Percentage of significantly positive correlations for different $N$ values

Fig. 4 shows variation trends of the $t$-wise coverage and Pearson’s $r$ with respect to $t$. Note that, for each $t$ and $N$, the $y$-axis value in the left plot is obtained by averaging the coverage over all 31 FMs, and that in the right plot is obtained by averaging $r$ values over all FMs for all settings of $N_b$. For each value of $N$, as shown in Fig. 4, the $t$-wise coverage declines as $t$ increases. The interpretation of this phenomenon could be easy. As $t$ goes up, the number of valid $t$-sets grows as well. For a fixed $N$, it is therefore getting more and more difficult to cover these $t$-sets. As a natural consequence, the $t$-wise coverage declines. In contrast, as observed in Fig. 4, Pearson’s $r$ increases along with $t$. That is to say, stronger correlations are adhered to larger $t$ (which implies that similarity-based testing is particularly useful for higher $t$). The explanation of the above phenomenon can be related to the following fact—variations of $t$-wise coverage yielded by random test suites increase synchronously with $t$. Taking $N = 100$ as an example, average standard deviations of $t$-wise coverage over all FMs are shown in Fig. 5. As clearly seen, they increase with $t$. Larger variations potentially lead to stronger correlations between the $t$-wise coverage and novelty scores. Let us demonstrate this using examples in Fig. 3. It can be observed that variations of the $t$-wise coverage become larger and larger as $t$ increases. For example, the $t$-wise coverage for $t = 2$ varies slightly on both FMs. In this case, the rise on novelty score does not lead to an obviously synchronous increase (or decrease) of the $t$-wise coverage. Therefore, the novelty score has a weak correlation with $t$-wise coverage. However, as variations increase with $t$, their correlations become stronger and stronger.

![Figure 4. Variation trends (grouped by $N$) of the $t$-wise coverage and Pearson’s $r$ with respect to $t$.](image)

Finally, we investigate how Pearson’s $r$ values are affected by different settings of $N_b$. As mentioned in Section 5.1.1, $N_b$ is an important parameter in our approach, and its (nearly) optimal value deserves careful explorations. For each pair of ($N$, $t$), as shown in Fig. 2, the mean Pearson’s $r$ (over all FMs) tends to increase with $N_b$, and then reaches the peak value around $N_b = N/2$. Finally, it tends to decline as $N_b$ is increased from $N/2$ to $N$. Notice that the worst $r$ values are obtained at $N_b = N$ in all the cases. This clearly suggests that, compared with the similarity-based fitness [4] (i.e., $N_b = N$), novelty scores with $N_b \neq N$ have stronger correlations with $t$-wise coverage, and therefore would be more suitable as a substitution of the $t$-wise coverage. Moreover, variation trends of Pearson’s $r$ indicate that $N/2$ could be a proper setting for $N_b$. Indeed, $N_b = N/2$ yields the largest $r$ values in most cases. It is worth mentioning that the above conclusions are drawn based on the mean Pearson’s $r$ over all FMs. Using $t$-wise coverage as another indicator, a fine-gain tuning of $N_b$ on individual FMs will be given in Section 5.5.2, where one will find that both indicators generally lead to similar conclusions.

**5.1.4 RQ1 summary**

The correlation analysis performed in this section brings out the following conclusions. First, novelty scores (including the similarity-based fitness) have, in most cases, a significantly positive correlation with the $t$-wise coverage. The above finding explains why novelty score can be used as a surrogate metric
for the t-wise coverage. Most importantly, it makes NS-based SPL testing theoretically solid as the improvement on novelty score will indirectly lead to a high t-wise coverage. Second, strengths of the correlations are affected by some key factors, e.g., the size of test suites (N), t-wise strengths and neighbor size (N_b). In general, stronger positive correlations are observed for smaller N and larger t. Moreover, novelty scores (with N_b = N) have stronger positive correlations with the t-wise coverage than the similarity-based fitness suggested by Henard et al. [4] (i.e., N_b = N_0). This implies that novelty scores (with N_b = N) could be more effective than the similarity-based fitness in guiding the search towards a set of configurations with decent t-wise coverage. In the forthcoming subsection, we will empirically verify this by comparing NS with four related approaches, one of which is the GA developed on the basis of the similarity-based fitness.

5.2 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 specify experiment setup used in the empirical study. In what follows, we give a brief introduction to the adopted statistical test tools. Finally, experimental results are presented and conclusions are summarized.

5.2.1 Algorithms under comparison

We compare NS with four state-of-the-art algorithms, i.e., Unpredictable [4], GA [4], SamplingDown [6] and Smarch [72]. Notice that we choose the first three 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 fourth algorithm Smarch is also included as it is the latest algorithm that achieves uniform sampling. The introduction to Unpredictable, GA and Smarch has already been presented in Section 3. Here we only give some notes on SamplingDown, which is adapted by us from the similarity-based prioritization [6]. The original approach, applied on any product sample, focuses solely on ordering products to increase the probability of detecting faults faster. In this section, we extend it to a sampling approach, in which a large initial set of configurations are generated in an unpredictable way, and then some promising configurations are selected one by one based on similarity in the same way as described in [6].

5.2.2 Experimental setup

Experiment setups used in this section are summarized as follows.

- The number of returned configurations is 100 for all algorithms, following the practice in [4].
- Each algorithm is independently run 100 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 (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 [49]. For Unpredictable, SamplingDown and Smarch, their termination can not be manually controlled. Instead, they are automatically terminated once 100 configurations are sampled. Note that, to mitigate the random bias, both NS and GA receive the same sample returned by Unpredictable as initial population for every run.
- In the NS algorithm, control parameters are set as follows: P_r = 0.1 and N_b = N/2. This setting for N_b is based on the correlation analysis presented in Section 5.1.3, while P_r is experimentally tuned with details to be given in Section 5.5.1. The GA algorithm is similar to NS, except for the following two major differences. First, GA adopts no repair operators. Second, it uses similarity-based fitness instead of a general novelty score. Therefore, using the same framework as in NS, GA can be implemented with the following parameter settings: P_r = 0 and N_b = N. This enables us to well demonstrate benefits brought by repair operators and novelty scores. Note that the Anti-dice distance is chosen as the distance metric in both NS and GA. To be fair, the number of initial configurations in SamplingDown is set to the total number of configurations explored during the whole search process of NS. For example, suppose that 2,000 configurations are explored in the search process of NS on some FM for some run, then the number of initial configurations in SamplingDown is set to 2,000. For Unpredictable and Smarch, no external control parameters are required.

5.2.3 Statistical test tools

To make comparisons easy, several statistical test tools are adopted in this section (and also in other related sections).

First, following the guidelines suggested by Arcuri and Briand [82], the Mann-Whitney U test with a 0.05 significance level is performed to determine whether the difference between NS and each of the peer algorithms (over all the runs) is significant or not. In particular, the symbols *, † and ‡ indicate that NS performs better than, equivalently to or worse than the corresponding algorithms, respectively. This significance test is applied to each possible combination of FMs and t values. Tables A-2, A-4 and A-5 in the appendix give mean values and standard deviations of the t-wise coverage obtained on small-scale, moderate- and large-scale FMs, respectively. In these tables, Mann-Whitney U test results are presented. In addition, the best mean result for each FM and each value of t is indicated by a dark-gray background, and the second-best result a light-gray background.

Second, the Friedman test13 [83] is adopted to compute average rankings of the algorithms considering all FMs and all t values (resulting in 31 × 5 = 155 cases in total). This test procedure, which has been widely employed to evaluate evolutionary algorithms [84], and search-based test case generators [85], first ranks the algorithms in each case separately and then computes an average ranking over all cases. Clearly, the smaller the ranking, the better the

13. The Friedman test is implemented by the KEEL Software Tool available at https://sci2s.ugr.es/keel/.
algorithm. The Friedman test is very useful to compare the overall performance of the algorithms.

Third, the Vargha and Delaney’s $A_{12}$ statistic\(^\text{14}\) [86] is utilized to evaluate the effect size, i.e., determine which algorithm leads to higher $t$-wise coverage and to what extent [82], [87]. This test returns Vargha and Delaney’s A effect size (or simply ES) and a qualitative assessment of the magnitude of ES (or simply magnitude). The value of the $A_{12}$ ES ranges from 0 to 1, and when the value is exactly 0.5, it means that the peer algorithm either provides exactly the same $t$-wise coverage as NS for all runs, or provides higher $t$-wise coverage than NS for exactly 50% of the runs. If the value of $A_{12}$ ES is lower than 0.5, the peer algorithm gives higher $t$-wise coverage than NS for a majority of the runs. Conversely, if the value of $A_{12}$ ES is higher than 0.5, NS provides higher $t$-wise coverage for a majority of the runs. According to Vargha and Delaney [86], magnitude of the effect size can be qualitatively assessed as ‘Large’, ‘Medium’, ‘Small’ and ‘Negligible’. The $A_{12}$ statistic is helpful to demonstrate whether or not NS has substantial improvements compared with the state-of-the-art algorithms. The $A_{12}$ statistical results for each pairwise comparison between NS and each of the other algorithms are listed in Tables A-6 to A-8 in the appendix.

### 5.2.4 Experiment results and discussions

According to the summaries of the best/second-best results, NS is the best-performing algorithm on small-scale FMs, obtaining the best/second-best results in about 59/60 $\approx 98\%$ cases. It is followed by GA and SamplingDown, both of which perform either best or second best in 32/60 $\approx 53\%$ cases. Finally, Unpredictable and Smarch, the two most ineffective algorithms, perform best/second best in only one and two cases, respectively. Regarding the efficiency, Fig. 6 shows that Unpredictable is the most efficient algorithm, taking only 0.03 seconds on average to generate 100 configurations. SamplingDown is slower than NS, requiring nearly triple running time consumed by NS. For Smarch, the slowest algorithm, the sampling takes 119 seconds on average, being nearly 20 times slower than NS. Considering the efficiency, Smarch is not competitive. In fact, according to [72], this algorithm can be expensive to scale to larger FMs. For example, Smarch failed to generate at least 12 configurations within an hour on 2.6.28.6-icse11 [72]. Therefore, it is not included in the performance evaluations on moderate and large FMs\(^\text{15}\).

Experimental results on moderate- and large-scale FMs show that NS is still the best algorithm, outperforming the other three algorithms concerning the number of the best/second-best results obtained. Going one step further, we compute average rankings of the algorithms by applying the Friedman test. Results in Table 3 clearly suggest that the performance of the algorithms follows the order: NS > GA > SamplingDown > Unpredictable.

\(^\text{14}\) The $A_{12}$ statistic is performed using the effsize package in R platform.

\(^\text{15}\) Another reason leads us to choose Unpredictable instead of Smarch as a baseline is that the two algorithms perform competitively to each other concerning both $t$-wise coverage and novelty scores (see Tables A-2 and A-3 in the appendix, respectively).

![Figure 6. Average running time on small-scale FMs. ‘SD’ and ‘UP’ stand for SamplingDown and Unpredictable, respectively.](image)

**Table 3**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS</td>
<td>1.1935</td>
</tr>
<tr>
<td>GA</td>
<td>2.1935</td>
</tr>
<tr>
<td>SamplingDown</td>
<td>2.6677</td>
</tr>
<tr>
<td>Unpredictable</td>
<td>3.9452</td>
</tr>
</tbody>
</table>

Table 4 summarizes the $A_{12}$ statistical results for each magnitude of ES. As seen, large differences are found in the majority of the cases. Precisely, compared with GA, SamplingDown and Unpredictable, large differences in favour of NS (i.e., $ES > 0.5$) are observed in 123, 130 and 148 out of 155 cases, respectively. Conversely, large differences in favour of the peers are detected in only 6, 5 and 0 cases for GA, SamplingDown and Unpredictable, respectively. To sum up, the $A_{12}$ statistic confirms the substantial superiority of NS over the other algorithms.

**Table 4**

<table>
<thead>
<tr>
<th></th>
<th>NS vs GA</th>
<th>SamplingDown</th>
<th>Unpredictable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>$ES &gt; 0.5$</td>
<td>123</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>$ES &lt; 0.5$</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Medium</td>
<td>$ES &gt; 0.5$</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>$ES &lt; 0.5$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Small</td>
<td>$ES &gt; 0.5$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>$ES &lt; 0.5$</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Negligible</td>
<td></td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

Since both NS and GA are search-based approaches, their termination conditions can be flexibly specified. One may be interested in how the performance (regarding the $t$-wise coverage) varies when the termination conditions are changed. In addition, one may be also interested in how NS performs in comparison with GA when $N$ is set to different values, other than 100. To this end, we run both NS and GA using half or double $max \_t$ (which is specified in Section 5.2.2). Note that $N$ in these runs is fixed to 100 in both algorithms. Moreover, the two algorithms are executed with $N$ being 50 or 500, and $max \_t$ being the original values as specified in Section 5.2.2. Mean values and standard deviations of the $t$-wise coverage obtained in each scenario are provided in Tables A-9 to A-14 in the appendix. Similarly, Mann-Whitney U tests are performed to determine whether the differences are significant or not, and the test results are summarized in Table 5, where we list the percentage of cases in which NS performs significantly better than (●), equivalently to (¶) and significantly worse.
than (>) GA. As seen, NS consistently improves over GA regardless of the settings of max_t and N. Indeed, the cases in which NS wins are (significantly) more than those it loses. Notice that NS and GA perform competitively in a considerable number of cases for N = 500, particularly on small and moderate FMs. One of the possible explanations of this would be that benefits of using a large population overwhelm those of search-based techniques. To be more specific, using a large population alone could be sufficient to cover decent t-wise combinations on small and moderate FMs. In such a case, the difference caused by different search techniques can be subtle. For large FMs, however, NS significantly outperforms GA. Due to colossal t-wise combinations for large FMs, there is a lot of room for t-wise coverage improvement even though N is as large as 500. In this case, the effects of different search approaches become evident. We must mention that the superiority of NS for relatively small N is particularly meaningful in practice. Indeed, due to limited test budgets [4], [8], software testers generally prefer small test suites that can cover t-wise combinations as many as possible.

<table>
<thead>
<tr>
<th>Table 5</th>
<th>Summary of Mann-Whitney U test results comparing NS against GA when max_t and N vary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Half time</td>
</tr>
<tr>
<td>N = 100</td>
<td>102/155 ≈ 66%</td>
</tr>
<tr>
<td>N = 200</td>
<td>32/155 ≈ 21%</td>
</tr>
<tr>
<td>N = 300</td>
<td>21/155 ≈ 14%</td>
</tr>
</tbody>
</table>

Finally, we compare NS against the original GA implemented by Henard et al. [4]. This version of implementation is named as Henard’s GA hereafter. Indeed, in all the previous experiments, GA is implemented by ourselves in the NS framework with specific parameter settings: Pf = 0 and N0 = N. In fact, Henard’s GA follows a general framework as in traditional evolutionary algorithms. It treats the whole test suite as an individual, and iteratively applies mutation to this individual to obtain a new one. Both individuals are evaluated based on the similarity-based fitness function defined in (5), and the new individual is accepted if the fitness value is increased. Note that Henard’s GA comes with two prioritization strategies, the local maximum distance and the global maximum distance [4]. In this implementation, we choose the local maximum distance. To be fair, N and max_t in both algorithms are kept identical, as specified in Section 5.2.2. It is worth mentioning that N0 in NS is set to N in this experiment because novelty score with this setting is equivalent to the fitness function used in Henard’s GA. This helps to get rid of the impacts of inconsistent performance measures. Moreover, both algorithms use Anti-dice distance to evaluate the degree of similarity between two products. Fig. 7 shows the t-wise coverage obtained by both algorithms in the form of boxplots (over 31 FMs for each t). As seen clearly, NS performs better than Henard’s GA regardless of the value of t. Notice that some outliers are detected, and they are mainly observed on Automotiv02 for which both algorithms have low t-wise coverage (because this model have 18,434 features, significantly larger than other models). Nevertheless, NS has slightly higher coverage than Henard’s GA for all t. For example, the 2-wise coverage is 78.310% and 78.253% for NS and Henard’s GA, respectively. Therefore, we can conclude that NS is generally more effective than Henard’s GA (with the local maximum distance), in which the mutation operator is adopted.

5.2.5 RQ2 summary
The following are some conclusions drawn from the above numerical results. First, the two search-based approaches, i.e., NS and GA, are in general more effective than the three sampling algorithms, i.e., SamplingDown, Unpredictable and Smarch. This observation can be explained by the results of the correlation analysis performed in Section 5.1. It has been shown that novelty scores used in NS and GA have a significantly positive correlation with the t-wise coverage in most cases. Therefore, the improvement on novelty scores will potentially increase the t-wise coverage. In the sampling algorithms, however, there are no explicit mechanisms being adopted to potentially improve of the t-wise coverage. Second, according to the Friedman test, NS obtains the best average ranking. Moreover, the A_12 statistic shows that differences between NS and the other algorithms are substantial, with a majority of the effect sizes being large. Third, NS shows a clear superiority over GA, and this superiority is generally insensitive to the termination conditions and the size of test suites. This observation is in line with the finding of the correlation analysis. It has been observed in Section 5.1 that the novelty score used in NS has (much) stronger positive correlations 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.

5.3 Evaluations of fault detection capability (RQ3)
In this section, we evaluate the algorithms in terms of the fault detection capability, using FMs with either real or simulated faults.

5.3.1 Evaluations on Drupal with real fault data
In this part, we use the Drupal framework (http://www.drupal.org) as a case study. Drupal is a widely used open source web content management system, whose feature model and real fault data are publicly available.

Figure 7. T-wise coverage obtained by NS and Henard’s GA, shown in boxplots
GA for t missed by Unpredictable for each t, except for only three test cases (i.e., N = 3, 11, 38, and 126, which are the sizes of the FMs). For a single feature (t = 1), it may contain multiple faults. We assume that all the faults identified in this feature can be detected if the feature is covered by a test suite. To evaluate the fault detection capability, we adopt fault detection rate as an indicator. It is defined as the ratio of the number of detected faults to that of all faults. Clearly, a high fault detection rate is desired.

<table>
<thead>
<tr>
<th>t value</th>
<th># faults</th>
<th># t-sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>322</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
<td>3,751</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>103,267</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2,065,754</td>
</tr>
</tbody>
</table>

**Experimental setup.** Experiments are performed following Fischer et al.’s work [33]. Specifically, four values for N are examined, i.e., 3, 11, 38, and 126, which are the sizes of the t-wise covering arrays calculated using CASA [16] for t = 1, 2, 3, and 4, respectively. In both NS and GA, max_f is set to 1 minute for N = 3 and N = 11, and to 10 minutes for N = 38 and N = 128. These settings are exactly the same as in [33].

<table>
<thead>
<tr>
<th>N</th>
<th>t</th>
<th>Fault detection rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>97.18%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>93.48%</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>76.67%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N, size</th>
<th>t</th>
<th>Fault detection rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>11, 38, 126</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>11, 38, 126</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>11, 38, 126</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>11, 38, 126</td>
<td>4</td>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>t</th>
<th>t-wise coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>99.77%</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>99.77%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>99.77%</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>99.77%</td>
</tr>
</tbody>
</table>

**5.3.2 Evaluations on FMs with simulated faults**

**Experimental setup.** For FMs without real fault data, we generate simulated faults using the fault simulator described by Al-Hajjaji et al. [6]. This simulator was originally presented by Bagueri et al. [88], and has been widely used to evaluate the fault detection rate of SPL test suites [62, 88, 89]. The input of the simulator is a feature model, and the outputs are valid combinations of features which cause faults. The steps of generating these simulated faults are as follows. First, a value t ∈ [1, 6] is randomly selected. Then, we randomly pick t features, with each feature being randomly decided whether it is selected or deselected. Finally, we check the validity of this combination of features using SAT solvers. Note that only valid combinations are accepted, because invalid ones are not relevant. The above steps are repeated until a certain number of faults are generated. Following the practice in [6], the number of simulated faults for each feature model is n/10, where n is the number of features. Clearly, this number is proportional to the feature model size. The rationale behind this is that larger feature models lead to potentially more feature interactions. To mitigate the impacts of randomness, 100 sets of simulated faults are generated for each feature model, and we take the mean fault detection rate (over these fault sets) for each run of an algorithm.

**Experimental results.** Table A-15 in the appendix presents average fault detection rates (over 100 runs) of the test suites returned by the algorithms. As shown clearly, the average rates fail to reach 100% for any algorithm on any FM, but are reasonably high in most cases. For example, more than 80% faults are detected by NS, GA and SamplingDown on the majority of the FMs. In particular, the largest rate is 99.966%, which is obtained by NS on toybox. According to the Mann-Whitney U test results presented at the bottom of that table, NS is the best performing algorithm regarding...
the fault detection capacity, significantly outperforming GA, SamplingDown and Unpredictable on 23, 26 and 30 out of 31 FMs, respectively. It can be found that the order of the algorithms concerning the fault detection capacity is in line with that concerning the t-wise coverage capacity presented in Section 5.2. This well verifies that test suites with higher t-wise coverage will potentially make more faults covered.

### 5.3.3 RQ3 summary

Evaluations of fault detection capacity on FMs with both real and artificial faults emphasize the following. First, similarity-based SPL testing approaches (e.g., NS, GA and SamplingDown) offer no guarantees on 100% fault detection rates, especially when the size of test suites is small. However, the fault detection rates can be reasonably high in most cases. In particular, all faults are successfully identified when 11 or more test cases are generated in the Drupal case study. Second, NS yields the highest fault detection rate in the majority of the cases, which is consistent with its performance concerning t-wise coverage. This confirms that improving t-wise coverage indeed enhances the fault detection capacity.

### 5.4 Usefulness of combining two types of SAT solvers (RQ4)

This part investigates whether it is useful to combine two types of SAT solvers when generating new configurations in search-based SPL testing. Currently, the most-widely used generation approach is realized by the randomized SAT4J solver [4]. It has been well demonstrated that combining two types of solvers performs better than using only one solver in the context of selecting optimal SPL products [49, 50]. However, it remains open whether this is also the case in this context. The answer to this question will be made clear in this section after a comparative study.

**Experimental setup.** Within NS, we empirically evaluate the following three strategies for generating new configurations. The first one, as described in Section 4.2 and currently being used, iteratively generates random configurations that are handled by two different types of SAT solvers in case of infeasibility: probSAT [52] for repair and randomized SAT4J for replacement. In the remainder of this paper, we refer to this strategy as Random+TwoSAT. The second one 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 [7, 32, 47, 48, 49, 50]. Choosing this strategy as a baseline enables us to investigate whether it is useful to employ two different SAT solvers. Finally, the third strategy follows a traditional way as in evolutionary algorithms. In this strategy, genetic operators (uniform crossover and single-point mutation) are applied to generate a temporary configuration. As in the first strategy, this configuration will be then handled by the same two SAT solvers. We refer to this strategy as Genetic+TwoSAT hereafter. Comparisons with this strategy allow us to explore whether genetic operators are necessary in cases where two SAT solvers are used.

**Experimental results.** Considering Table A-16 in the appendix, it is found that differences between Random+TwoSAT and Randomized SAT4J are not statistically significant on 4 out of the 6 large-scale FMs regardless of the value of $t$. However, Random+TwoSAT shows a significant improvement over Randomized SAT4J on freebsd-icse11 and 2.6.28.6-icse11 with respect to all values of $t$. This clearly confirms the usefulness of combining two SAT solvers on these FMs. By comparing Random+TwoSAT and Genetic+TwoSAT, one can find that genetic operators perform statistically equivalently to the simple randomization method when generating new configurations. Fig. 8 intuitively shows that Random+TwoSAT and Genetic+TwoSAT have similar performance, and both outperform Randomized SAT4J in terms of the median of the t-wise coverage for all $t$ values.

![Figure 8. T-wise coverage obtained by the three generation strategies on six large-scale FMs, shown in boxplots](image)

**RQ4 summary.** Experiments performed in this section emphasize the following. First, the combination of two SAT solvers is useful, and therefore 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 [49]. 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 [49]. In this paper, we show, for the first time, this is also the case for the similarity-based testing of SPLs. In fact, using two solvers offers two completely different (greedy vs. systematical) manners in which the configuration space is explored. This may further increase diversity of the generated configurations, and it potentially leads to high performance in terms of t-wise coverage. Moreover, when aided by two SAT solvers, the simple randomization method (which is used to generate temporary configurations as input of the two SAT solvers) can be as effective as genetic operators. The former, however, is quite easy to use because it requires no efforts of tuning control parameters, like the crossover and mutation probabilities involved in genetic operators. This is a great benefit brought by using two SAT solvers.

### 5.5 Effect of key parameters in NS (RQ5)

From the practical point of view, it is important to provide some suggestions on settings of the key parameters involved in the NS algorithm. As shown in Algorithm 1, repair probability ($P_r$) and neighbor size ($N_b$) are two key parameters that need to be carefully tuned. Given limited spaces, the parameter tuning results for $P_r$ and $N_b$ are given in Figs. A-1&A-2, and Figs. A-3&A-4 in the appendix, respectively.
5.5.1 Sensitivity analysis on $P_r$

The $P_r \in [0, 1]$ determines the probability of using probSAT solver. To tune $P_r$, $N_b$ is fixed to $N/2$. Under this setting, we then change $P_r$ from 0.0 to 1.0 with a step size 0.1, and present the t-wise coverage over 100 runs in the form of boxplots for each value of $P_r$. Moreover, a green line is used to connect two adjacent average values. By doing so, it makes easy observing how the average t-wise coverage changes as $P_r$ increases. Note that, to eliminate the influence of initial population, the same configurations are initialized for all values of $P_r$ in each independent run.

As observed from our experimental results, the t-wise coverage for all values of $t$ slightly fluctuates as $P_r$ increases from 0.0 to 1.0 on the two small FMs (i.e., WebPortal and CocheEcologico), and tends to descend on the two moderate FMs (i.e., E-shop and axTLS), especially on E-shop. This observation suggests that the optimal value of $P_r$ should not be too large. Considering the large FMs (i.e., freedsacticc11 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 tends to decrease for $P_r \geq 0.1$. This implies that $P_r = 0$ is definitely not a good setting for $P_r$. In fact, $P_r = 0$ behaves the same as the Randomized SAT4J in Section 5.4, where we have already well demonstrated that using two SAT solvers performs better than Randomized SAT4J. Moreover, we find that, when the maximum running time ($\text{max}_t$) is doubled, similar trends are observed on the same FM, regardless of the value of $t$. This indicates that termination conditions have subtle impacts on these trends. Considering all the above scenarios, it would be reasonable to set $P_r$ to 0.1 for a common use.

5.5.2 Sensitivity analysis on $N_b$

As mentioned previously, $N_b$ is an important parameter that determines how many individuals in a population are used to compute the novelty score. Clearly, $N_b$ is directly related to the population size, $N$. To tune this parameter, we fix $P_r$ to 0.1. Then, we let $N_b$ be proportional to $N$, i.e., $N_b = \alpha N$, where $\alpha \in (0, 1]$. In a similar way as in the previous subsection, sensitivity analyses are conducted by examining typical values for $\alpha$ on several FMs. More specifically, we change $\alpha$ from 0.1 to 1.0 with a step size 0.1. To reduce the random bias, as done in tuning $P_r$, the same initial population (or test suite) is used for varying values of $\alpha$ in each independent run.

For each chosen FM and each value of $\alpha$, the t-wise coverage over 100 runs is exhibited in the form of boxplots. From these plots, we identify the following two types of curves.

- Type-I curves, as observed on FMs from Counter-StrikeSimpleFM to CocheEcologico (note that the order is from left to right, and from top to bottom), rise as $\alpha$ increases from 0.1, and quickly reach a peak. They then decline until $\alpha$ reaches a large value. Finally, the curves rise, in general, again till $\alpha = 1.0$. The optimal value of $\alpha$ for this type of curves tends to be small. It is roughly in the range $[0.3, 0.4]$ for $N = 50$, and around 0.2 for $N = 100$. This type of curves is mainly observed on small-scale FMs.
- Type-II curves, as observed on the remaining FMs, are unimodal. The only peak is obtained when $\alpha$ takes a moderate value within its range. We find that $\alpha \in [0.4, 0.5]$ shows promising performance in most of the cases. This type of curves is observed on small-, moderate- and large-scale FMs.

In general, the above observations indicate that $\alpha$ should not be too large (over 0.8), or too small (below 0.2). In practice, since Type-II curves are more common than Type-I curves, one can set $\alpha$ to 0.4 or 0.5 for a common use. That is, $N_b = 0.4N$ or $N_b = 0.5N$ is recommended. In fact, this setting is consistent with that tuned based on the correction analysis presented in Section 5.1.

5.5.3 RQ5 summary

Experiments performed in this section bring out the following outcomes. First, the recommended value for $P_r$ is 0.1. Under this parameter setting, NS shows promising performance on small-, moderate- and large-scale FMs. Moreover, this recommendation is independent of the termination conditions. Second, the (nearly) optimal value of the parameter $N_b$ can be 0.4N or 0.5N. It is also found that this suggested setting is independent of the population size $N$.

6 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 publicly available\(^\text{17}\) to enable reproducibility and to reduce the aforementioned threats.

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 100 times, and compare the performance based on means and standard deviations. In addition, to make comparisons reliable, non-parametric statistical tests are applied to determine whether the differences between algorithms are significant or not. Moreover, effect sizes are also computed to identify if these differences are substantial.

Internal validity threats may also be 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 or exactly generated) to compute the t-wise coverage of

\(^{17}\) https://github.com/gzhuxiangyi/TSE_NS
the configurations returned by each algorithm. This way, it enables a relative performance comparison among the algorithms. In addition, we make the number of randomly sampled \( t \)-sets relatively high, and also increase with the value of \( t \). To reduce bias caused by randomness, 10 sampling sets are generated for each FM and each value of \( t \). The \( t \)-wise coverage is then averaged over these sampling sets. Similar strategies are adopted to mitigate the impacts of randomness when generating simulated faults.

This kind of threats could also exist in the parameter tuning phase. For example, different initial test suites may overwhelm the impact of different parameter settings. To eliminate the influence of initial population, the same test suite is initialized for varying parameter settings in each independent run.

**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 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 [26], [27], [28], 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.

## 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 testing 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 similarity-based testing of SPLs. It is meaningful to exploit potential benefits of more suitable/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 prove that the novelty score used in NS is a generalization of the state-of-the-art similarity-based fitness. Further, we show, after performing a correlation analysis, that both similarity metrics have significantly positive correlations with \( t \)-wise coverage. This well explains the rationale of similarity-based testing. Moreover, we give detailed discussions on the suitability of NS for similarity-based SPL testing. The suitability of NS and its superiority over some state-of-the-art approaches are demonstrated through comparative studies performed on 31 feature models, either realistic or artificial, concerning the capacity of both covering \( t \)-wise combinations and detecting faults. In particular, it is found that NS significantly outperforms the state-of-the-art GA [4]. This observation conforms to the correlation analysis outcome, stating that novelty score used in NS has stronger positive correlations with \( t \)-wise coverage than the similarity-based fitness adopted by GA.

Next, we show that combing two SAT solvers is useful when generating new configurations in the context of similarity-based SPL testing. Finally, sensitivity studies are performed to find (nearly) optimal settings for key control parameters involved in NS. In particular, we suggest to set the neighbor size \( N_2 \) to around half of the population size.

Our results show that NS is promising for similarity-based testing of SPLs. In the future, it is possible to integrate objectives functions [7], 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 a test suite, without explicitly considering prioritization. Developing NS-based techniques that 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 [90], and behavioural SPL testing [39].

## Acknowledgments

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

## References


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 design 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.

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 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.