Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-wise Test Configurations for Software Product Lines

Christopher Henard, Mike Papadakis, *Member, IEEE*, Gilles Perrouin, *Member, IEEE*, Jacques Klein, *Member, IEEE*, Patrick Heymans, *Member, IEEE,* and Yves Le Traon, *Member, IEEE.*

Abstract—Large Software Product Lines (SPLs) are common in industry, thus introducing the need of practical solutions to test them. To this end, t-wise can help to drastically reduce the number of product configurations to test. Current t-wise approaches for SPLs are restricted to small values of t. In addition, these techniques fail at providing means to finely control the configuration process. In view of this, means for automatically generating and prioritizing product configurations for large SPLs are required. This paper proposes (a) a search-based approach capable of generating product configurations for large SPLs, forming a scalable and flexible alternative to current techniques and (b) prioritization algorithms for any set of product configurations. Both these techniques employ a similarity heuristic. The ability of the proposed techniques is assessed in an empirical study through a comparison with state of the art tools. The comparison focuses on both the product configuration generation and the prioritization aspects. The results demonstrate that existing t-wise tools and prioritization techniques fail to handle large SPLs. On the contrary, the proposed techniques are both effective and scalable. Additionally, the experiments show that the similarity heuristic can be used as a viable alternative to t-wise.

Index Terms—Software Product Lines, Testing, T-wise Interactions, Search-based Approaches, Prioritization, Similarity

1 Introduction

A *Software Product Line* (SPL) "is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [1]. Features are thus the key to the discrimination of SPL members by showing their commonalities and differences. A feature represents the presence of a functionality in a software product. These functionalities are often organized in a Feature Model (FM) [2], [3] which represents all the possible (software) products of the SPL by expressing relationships and constraints between features.

Testing an SPL is an inherently difficult activity [4]. Although testing all the products would be ideal, it is rarely feasible in practice. Indeed, the number of possible product configurations induced by a given FM usually grows exponentially with the number of features, quickly leading to millions of possible products to test. As a result, test engineers are seeking for solutions to reduce the size of their test suites so that they can meet release deadlines and cost constraints.

Previous work [5], [6] has identified Combinatorial Interaction Testing (CIT) as a relevant approach to reduce the size of the test suites using interaction coverage. CIT is a systematic approach for sampling large domains of test data. It is based on the observation that most of the faults are triggered by the interactions between a small number of features [6]. An interaction between $t \geq 2$ features denotes the possible impact of one feature on the others, enabled in a specific product configuration. Kuhn *et al.* [6] have shown that interactions between two features are able to disclose 80% of the bugs. Considering all possible interactions between t features is called t-wise testing. Recently, such approaches have been adapted to SPL testing [7], [8], [9], generating product configurations from the FM covering all the valid (with respect to the FM constraints) t-wise combinations of features.

However, computing all the t-wise interactions in the presence of constraints, as it is the case for FMs, is a hard problem to solve [9], [10]. Although t-wise generation techniques from FMs have been greatly improved over the last years, they still face scalability issues in the presence of constraints [11], [12], [13]. Therefore, dealing with large FMs such as those used in industry is still an open research issue. Furthermore, the CIT literature points out the need for dealing with higher interaction strengths ($t > 2$) [14], [15], [16]. Preliminary evidence shows that 3-wise

- *C. Henard, M. Papadakis, J. Klein and Y. Le Traon are with the Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg.*
  E-mail: firstname.lastname@uni.lu
- *Gilles Perrouin is an FNRS postdoctoral researcher. Gilles Perrouin and Patrick Heymans are with the Precise Research Center In Software Engineering, University of Namur, Belgium.*
  E-mail: firstname.lastname@unamur.be

interactions may commonly appear in SPL testing practice [17] and that higher interaction strengths are important in achieving a higher fault detection [16]. Additionally, the results of the present study show that state of the art CIT tools fail to scale even on FMs of moderate size for high interaction strengths ($t = 3, 4$). Since such strengths may remain out of reach, one may ask if it is possible to cope with these difficult situations by relaxing the t-wise criterion. This leads us to the question of whether *we can mimic t-wise product configurations generation, partially but efficiently while achieving decent coverage?*

While t-wise testing drastically reduces the number of product configurations to consider, this number may still be too high to fit the budget allocated for SPL testing. For example, 2-wise coverage for the Linux FM (over 6,000 features) already requires 480 product configurations to be tested [18]. This observation is in line with Song *et al.* 's [19] motivation : since they reported that key interactions may involve up to 7 option settings, full 7-wise coverage of a realistic system (admitting such a computation is possible) may yield too many configurations to consider. Therefore, being able to prioritize product configurations is critical from a practical point of view.

The work presented in this paper is motivated by the results of Arcuri and Briand [11] who showed that, in the case of large models, random testing is competitive with CIT for finding interaction faults. Hence, they suggest the use of random testing as a possible way to circumvent the scalability issues of the CIT approaches. Unfortunately, real world applications involve constraints between features and the results of Arcuri and Briand do not hold when constraints are present.

Going one step further, we propose two approaches working with constraints capable of generating and prioritizing product configurations: (a) a randomized approach, named here as unpredictable, and (b) a search-based technique. Following the lines of Arcuri and Briand, we empirically investigate the probability of finding an interaction failure for $t = 2, ..., 6$. Our search-based approach efficiently generates valid product configurations, i.e., respecting the constraints of the FM, for t-wise testing. The innovative part of the proposed approach is that it is independent of t and able to operate on large and constrained models by both generating and prioritizing configurations.

Flexibility is required to meet a given testing budget especially for such systems [20]. Generally, our approach introduces flexibility in the testing process: the number of product configurations that can be tested, i.e., fitting the budget can be specified as well as the time allowed for generating them. Flexibility is backed by the design of our approaches that avoid direct computation of t-wise interactions by using product configurations dissimilarity measures.

The applicability of the proposed strategies is evaluated on both real and generated FMs. Our approach stands up against the comparison with existing tools for small and moderate FMs, while it is able to scale well to models with $6,000$ features for t up to 6. The experimental data and the implementation are publicly available at:

http://research.henard.net/SPL/.

In summary, the present paper provides the following insights:

1) We show that state of the art tools face severe scalability issues with large SPLs.
2) We propose a randomized and a search-based approach able to generate valid product configurations for t-wise testing for large SPLs.
3) We introduce two scalable product configuration prioritization techniques.
4) We perform a wide empirical study including FMs that contain more than 6,000 features.

The remainder of this paper is organized as follows: Sections 2 and 3 respectively present the context of this work and an example. Section 4 introduces the heuristic used by the approaches. Sections 5 and 6 respectively detail the configuration generation and prioritization techniques. Section 7 reports on the empirical study. Finally, Section 8 discusses the proposed approaches before Section 9 concludes the paper.

2 CONTEXT

This section first presents background concepts and notations used in this paper. Then, SPL testing and CIT are presented. Finally, work related to the present one is discussed.

2.1 Background

This section introduces testing and coverage for Software Product Lines (SPLs) through some concepts and definitions used in this paper.

2.1.1 SPL Products as Test Configurations

This work focuses on model-based testing of SPLs where the variability model is a *Feature Model* (FM). An FM encompasses the constraints linking the features. Feature modeling is a popular way to model SPL variability and it is by far the most reported in industry [21]. Thus, basing an SPL testing technique on FMs as means of documenting variability seems reasonable. Moreover, FMs may be used to reason about systems that are not SPLs according to the classical definitions [1], [22]. Thanks to FM reverse-engineering techniques (e.g., [23], [24]), one can apply SPL testing techniques to variability-intensive (or configurable) systems such as the Linux kernel [24] or Eclipse [25]. Thus, an FM can represent the variability of an SPL

or of a highly configurable system, such as the Linux kernel. In fact, an SPL is a highly configurable system with a very early feature binding [26].

Definition 1 (Feature Model (FM)): In this paper, we see an FM as a tuple (F, C), where $F = \{f_1, ..., f_n\}$ is set of n boolean features and $C = \{c_1, ..., c_k\}$ a set of k constraints over the features. C is satisfied if all its constraints are satisfied, i.e., evaluated to $true$.

In this context, we consider a product configuration as an assignment of selected/unselected features satisfying the constraints of the FM.

Definition 2 (Product Configuration): A product configuration is a set $PC = \{\pm f_1, ..., \pm f_n\}$, where $+f_i$ indicates a feature of the FM which is present in this product configuration, and $-f_i$ an absent one. A product configuration is said to be *valid* if C is satisfied, i.e., all the constraints of the FM are satisfied. Otherwise, it is said to be *invalid*. For simplicity reasons, we also refer to product configuration as configuration.

A configuration may actually refer to different things depending on the source of the features. While in the SPL terminology it usually refers to the configuration of a product of the SPL, where each of its features may have a complex behavior, it may as well represent parameter values if the FM is a configurable system, e.g., the Linux kernel. This is not a problem in our case since the approach described here is agnostic of feature semantics [27]. Regarding the SPL testing process, these configurations need to be embodied and relevant test cases for these configurations have be provided so that actual testing can be performed. The full process is out of the scope of this paper. Nevertheless, model-based product configuration derivation techniques show that this scenario is realistic even for large systems [25]. Finally, we denote as test configuration suite a list of (product) configurations.

Definition 3 (Test Configuration Suite): A test configuration suite is a list $TCS = PC_1, ..., PC_m$ where each PC_i is a valid product configuration.

2.1.2 T-wise Testing and Coverage

T-wise testing focuses on the interactions between any $t \geq 2$ features of an SPL [10]. It considers all the possible interactions (with respect to the constraints C of the FM) between selected and unselected features. Such an interaction is called a t-set.

Definition 4 (valid t-set): A valid t-set is a set $\{\pm f_1, ..., \pm f_t\}$ satisfying C, with $t \leq n$ and where $+f_i$ indicates a feature which is selected and $-f_i$ an unselected one. A t-set which is not satisfying C is said to be invalid.

Definition 5 (t-wise coverage): The t-wise coverage of a test configuration suite $TCS = PC_1, ..., PC_m$ is the ratio $\frac{\#\bigcup_{i=1}^{m} T_{t,PC_i}}{\#T_{t,FM}}$, where T_{t,PC_i} is the set of t-sets covered by the product configuration PC_i (i.e., t-sets included within the configuration PC_i), where $T_{t,FM}$ denotes the set of all the possible t-sets of the FM and where $\#A$ denotes the cardinality of the set A.

Classical approaches [7], [8], [9] to t-wise testing have coverage ratio of 1 as they cover all the t-sets of the FM. Finally, set coverage redundancy expresses the possibility that, by removing any configuration, the coverage value is not altered.

2.2 Software Product Line Testing and Combinatorial Interaction Testing

CIT aims at sampling configurations in order to reduce the size of the test suites. This reduction is achieved by keeping only the configurations covering all the interactions between t features. As a result, the strength t is a parameter of a CIT approach. CIT approaches are closely related to the configuration generation in SPLs. Generally, CIT handles multi-valued variables while configuration generation for SPLs limits the values of variables, i.e., the features, to boolean ones. In that sense, the generation of configurations in SPLs can be seen as a subset of CIT. However, constraints among variables are generally not included in CIT problems. Indeed, initial CIT approaches did not take into account constraints. Some recent studies, e.g., [26] evaluate the impact of constraints on CIT. Recent CIT tools provide a support of constraints. In other words, we can say that configuration generation in SPL testing is an instance of a boolean CIT problem with constraints.

Generally, an SPL configuration generation problem can be solved by a CIT tool if it handles constraints. Indeed, CIT with constraints generalizes the SPL configuration generation problem since it deals with multi-valued variables. However, to convert a CIT problem to an SPL one, it is necessary to transform the problem into a boolean one. If the variables' domain is finite, this transformation can be performed by applying the rules presented by Frisch *et al.* [28].

In the CIT context, the constraints have a great influence: they can, for instance, make configurations invalid, increase or decrease the number of configurations required to cover all the t-wise interactions, or produce invalid t-sets [26]. Thus, introducing constraints into a CIT problem makes it extremely difficult to solve. This is due to the irregularity introduced by the constraints [26]. The configuration generation in SPLs suffers from the same problems.

State of the art tools for solving CIT with constraints have great difficulties to scale to large FMs and to deal with high t values [11], [12], [13]. These issues are highlighted by our evaluations reported in the present paper (see Section 7.1). Indeed, none of the three state of the art tools employed by our study achieved to scale to large FMs. In addition, these tools fail also on moderate size FMs for t values greater than 3. As a result, scalability is one of the major problems faced by existing techniques. Additionally, due to the irregularity of the constraints, generating valid configurations at random is practically infeasible. The example of the next section will highlight this

problem. Therefore, there is a practical need to deal with situations involving both constrained large scale systems and high interaction strengths.

2.3 Related Work

Since t-wise testing is difficult due to the constraints, the use of constraint solving solutions have been investigated. In [26], Cohen *et al.* examine the impacts of constraints and present techniques to integrate constraints handling into existing CIT tools. In Perrouin *et al.* work [10], a solution based on Alloy, a satisfiability (SAT) solver, was devised. The approach was non-predictable in terms of generated solutions and strategies to improve scalability were proposed. Oster *et al.* [7] optimized the problem upfront by flattening the FM and using CIT algorithms [5], [29] within a dedicated constraint solver, producing predictable solutions. Both cannot handle thousand-sized FMs.

Recently, SPLCAT [9], used as a reference throughout this paper has been proposed. SPLCAT operates by generating a covering array [26]. In a covering array, the rows represent the product configurations while the columns represent the features. The approach is incremental and adds configurations (rows) until all the feature combinations are covered. Each configuration added in the array tries to exercise the maximum number of interactions that remain to be covered. This is performed using a SAT solver which, given assumptions representing the interactions to be covered, returns a product configuration. Configurations are added until all the interactions of the FM are covered. Our generation technique tries to maximize the dissimilarity of set of n configurations, where n is predefined. Thus, SPLCAT generate all the configurations needed to achieve 100% of t-wise coverage. On the contrary, our approach aims at maximizing the t-wise coverage of the n configurations. SPLCAT handles larger FMs than the other techniques, but it does not scale well. An improvement of SPLCAT has been recently proposed [18]: it handles larger FMs than SPLCAT but it is limited to $t = 3$.

Logic was also used. Calvagna *et al.* explain how to deal with constraints in CIT [30] by encoding them in first order logic. They offer various reductions algorithms to simplify them and used a model checker to solve them. Since this work was not related to FMs, it is difficult to assess its scalability. Hervieu *et al.* [31] also use reduction techniques in the aim of finding the minimal test suite in a Prolog-based implementation. However, this approach does not scale well to FMs of over 200 features, according to our experiments.

Cohen *et al.* [32] propose a relational model to represent the semantic basis for defining a family of coverage criteria for testing an SPL, such as variability coverage. CIT is then used to generate configurations that achieve a desired level of coverage. In our work, we focus on the notion of t-wise interaction coverage while generating configurations. The testing of the SPL itself is not considered. Finally, characterizing the features combinations responsible for failures can be performed with classification trees [33]. This is part of debugging and thus falls out of the scope of the present paper. Our paper focuses on the generation of product configurations in SPL testing.

As surveyed by Nie and Leung [13], efforts have been made to prioritize test suites. For instance, Bryce and Colbourn [34] use search-based techniques (e.g., hill climbing) to select the "best test" in terms of t-wise coverage. Our goal is similar but we focus on product configurations. Additionally, the proposed techniques offer improvements over the "natural" ordering provided by the AETG algorithm [5] in line with our experimentations. However, computing t-wise coverage for each configuration is expensive, especially for constrained cases, which are not taken into account in their approach and thus unsuitable in the SPL context. Yoo *et al.* [35] introduced a cluster-based prioritization technique to reduce the number of 2-wise interactions. The idea is too regroup similar test cases into clusters, and prioritize the clusters. In our work, we use a notion of similarity to compare test configurations. Prioritization of test configurations according to parameter interactions has also been done by Sampath *et al.* [36] in the context of web applications. Bryce and Memon [37] proposed a technique to prioritize configurations according to the t-wise interactions covered. It is a greedy approach which selects the configurations exercising the maximum number of interactions that are not already covered. The difference is that our approaches are not impacted by the t-wise interactions since they are independent of t. Indeed, our techniques select the most dissimilar configurations instead of those covering the highest number of interactions. Other work [38] also adds the notion of cost of the test to the combinatorial interaction coverage metric. In our work, we focus only on the t-wise interactions, assuming that all the configurations have the same cost. Finally, there are SPL-dedicated efforts, also in the context of test generation, but not directed to t-wise, such as Uzuncaova *et al.* [39] work.

Due to the computational complexity of t-wise testing of SPLs, using search-based heuristics is an option. However, we are only aware of two approaches [40], [41]. Garvin *et al.* [40] report on their experience applying and improving an extension to the AETG algorithm [5] using simulated annealing. The simulated annealing approach incrementally populates a *constrained covering array* [26]. It can be simply viewed as a table where lines represent configurations and columns features. Each change to the value of the features is controlled by a SAT solver to ensure it is legal with respect to the FM constraints. Changes are guided by a fitness function defined over the remaining pairs to be covered: the fewer pairs to be

covered, the lower the probability to make a change.

As it is shown in Section 5.1, using t-wise coverage as a fitness function induces scalability issues which may be intractable for very large FMs or high t values. Similarly to ours, Ensan *et al.* devised a genetic algorithm approach to generate SPL test configuration suites [41]. They propose an approach where each gene is a feature to be mutated and where crossover is applied. The crossover induces possible invalid products which need to be removed and thus they face scalability issues. Their fitness function indirectly measures coverage by evaluating the variability points to be bound and the constraints concerned by the features of a configuration. On the contrary, our approach copes better with large FMs ([41] does not scale over 300 features) and does not produce invalid configurations (since a configuration is always replaced as a whole). As opposed to other approaches, Ensan *et al.* and our approaches yield partial t-wise coverage due to the choice of the fitness function. This, however, allows dealing with time and cost constraints, looking for a "good enough" solution. Rubenstein *et al.* [42] proposes an algorithm that performs a search until some point in time in the context of software systems analysis. They use a measure for the accuracy of the analysis, which is also used to decide when to stop the process. This measure can be seen as the fitness function in our work. The difference with our approach is that the fitness function only evaluates the quality of the proposed solution, but does not indicate when to stop the algorithm. In our context, the stopping criteria is the time budget allowed.

Finally, the variations in space (different product configurations one can form from the FM) but also in time (product versioning) have been investigated in some work [43], [44]. In this work, we focus only on configuration space variations since we only consider one version of the SPL. Our approach can be adapted in the context of different version of an SPL by focusing only on the features that changed from one version to the other. Section 8.3 gives more details about the adaptation of our approach to SPLs evolving over time. Surveys [45], [46] report that SPL testing goes beyond the configuration generation. We do not strive to cover the full SPL testing process. Indeed, we focus on scalable product configuration generation, an open research issue [11], [12], [13].

3 Example

As a running example, consider a Raster Graphics Editor (RGE) SPL depicted by the FM of Figure 1. It contains 9 features that are mapped as follows: $RasterGraphicsEditor \mapsto f_1$, $Draw \mapsto f_2$, $Selection \mapsto f_3$, $ColorPalette \mapsto f_4$, $Rendering \mapsto f_5$, $Rectangular \mapsto f_6$, $ByColor \mapsto f_7$, $BlackWhite \mapsto f_8$, $Color \mapsto f_9$.

The RGE FM is defined by its 9 features and its 18 constraints[1] as (F, C), where $F = \{f_1, ..., f_9\}$ and $C = \{+f_1, -f_2 \vee +f_1, -f_1 \vee +f_2, -f_3 \vee +f_1, -f_1 \vee +f_3, -f_4 \vee +f_1, -f_5 \vee +f_1, -f_1 \vee +f_5, -f_6 \vee +f_3, -f_7 \vee +f_3, -f_3 \vee +f_6 \vee +f_7, -f_8 \vee +f_5, -f_9 \vee +f_5, -f_5 \vee +f_8 \vee +f_9, -f_8 \vee -f_9, -f_7 \vee +f_4, -f_4 \vee -f_8, -f_9 \vee +f_4\}$.

In the absence of constraints, $2^9 = 512$ product configurations can be established (9 features, with 2 possible values per feature). However, there are product configurations among these 512 that are invalid with respect to the constraints. Taking into accounts the constraints drops the number of configurations to 4 only (i.e., this SPL supports only 4 products). In other words, only 4 configurations among the 512 possible are valid ones. It means that, by trying to randomly generate a configuration, the probability to obtain a valid one is only $4/512 = 0.78\%$. As a result, generating valid configurations at random is rather unlikely, even for small SPLs. Thus, a systematic way to deal with valid configurations is in need. To deal with this situation, a SAT solver [47] to handle the constraints of the FM is mandatory. For the FM of Figure 1, the 4 configurations satisfying C that can be configured are the following:

$$PC_1 = \{+f_1, +f_2, +f_3, +f_4, +f_5, -f_6, +f_7, -f_8, +f_9\},$$
$$PC_2 = \{+f_1, +f_2, +f_3, +f_4, +f_5, +f_6, -f_7, -f_8, +f_9\},$$
$$PC_3 = \{+f_1, +f_2, +f_3, +f_4, +f_5, +f_6, +f_7, -f_8, +f_9\},$$
$$PC_4 = \{+f_1, +f_2, +f_3, -f_4, +f_5, +f_6, -f_7, +f_8, -f_9\}.$$

The t-sets of this FM can be computed as follows. Compute all the possible t combinations from $\{+f_1, ..., +f_9, -f_1, ..., -f_9\}$. Then, remove the combinations that are invalid. An example of valid 3-set is $\{+f_1, +f_2, -f_8\}$. An invalid 2-set is for instance $\{-f_1, f_2\}$, as it does not satisfies the constraint $-f_2 \vee +f_1$. Thus, the example FM encompasses 73 valid 2-sets, 204 valid 3-sets, etc. PC_1 and PC_4 together cover $66/73 \approx 90.4\%$ of these 2-sets and $\approx 80.4\%$ of the 3-sets. On the contrary, PC_2 and PC_3 together cover only $\approx 60.3\%$ of the 2-sets and $\approx 54.9\%$ of the 3-sets.

The objective of the *configuration generation* ap-

1. In this example, the constraints are represented in the Conjunctive Normal Form (CNF).

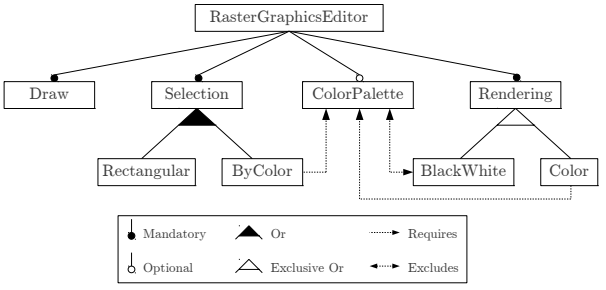


Fig. 1. A feature model of a raster graphics editor software product line. It depicts the 9 features and the constraints linking them.

proach presented in this paper is to, given a number of m configurations desired and an execution time, provide m valid configurations that maximize the t-wise coverage for any t value. In this example, if $m = 2$ (i.e., 2 configurations have to be generated), we expect them to be those that provide the maximum coverage. Therefore, PC_1 and PC_4 should be chosen rather than PC_2 and PC_3, as they cover more t-sets.

Suppose now that we have the three configurations PC_1, PC_2 and PC_3. All together, they provide a 2-wise coverage of $\approx 71.2\%$. PC_1 alone provides a 2-wise coverage of $\approx 49.3\%$. If we now consider PC_2 in addition to PC_1, it increases the coverage to $\approx 69.8\%$. However, if we consider PC_3 in addition to PC_1, the coverage is extended to only $\approx 60.2\%$. This difference is depicted in Figure 2. In other words, the order in which the configurations are considered allows reaching faster or slower the total coverage of $\approx 71.2\%$ provided by these configurations. In this case, it is more interesting to consider the order PC_1, PC_2 and PC_3 rather than the order PC_1, PC_3 and PC_2.

The objective of the *configuration prioritization* approaches presented in this paper is to, given a set of m configurations, order them such as the t-wise coverage provided by these configurations is achieved faster.

The following section details the concept of similarity underlying the proposed generation and prioritization approaches.

4 The Similarity Heuristic

Similarity is a heuristic used here to compare two configurations. In model-based testing, it has been found that dissimilar test suites have a higher fault detection power than similar ones [48]. The results presented in this paper (see Section 7) suggest that two dissimilar configurations are more likely to cover a greater number of valid t-sets than two similar ones.

In this context, we define a distance measure d between two configurations to evaluate their degree of similarity. As explained in Section 2.1.1, a configuration is considered as a set of selected or unselected features. Thus, a straightforward distance measure is a set-based one, like the Jaccard distance [49] or any other set-based distance metrics such as the Dice or Anti Dice measures [48]. If PC represents all the possible product configurations of an FM, the Jaccard distance is mathematically given by:

$$d: \begin{array}{ccc} PC \times PC & \longrightarrow & [0, 1.0] \\ (PC_i, PC_j) & \longmapsto & 1 - \frac{\#PC_i \cap PC_j}{\#PC_i \cup PC_j}. \end{array}$$

The resulting distance varies between 0 and 1. More particularly, a distance equal to 1 indicates that the two considered configurations are completely different. A distance equal to 0 denotes that the two configurations are the same (redundant). It is noted that an unselected feature is also an element of the set representing a configuration. For instance, with reference to the example of Section 3, $d(PC_1, PC_2) = 1 - \frac{\#\{+f_1,+f_2,+f_3,+f_4,+f_5,-f_8,+f_9\}}{\#\{+f_1,+f_2,+f_3,+f_4,+f_5,-f_6,+f_6,-f_7,+f_7,-f_8,+f_9\}} = 1 - \frac{7}{11} \approx 0.36$ and $d(PC_1, PC_4) \approx 0.71$. In this example, PC_1 and PC_2 are the most similar configurations (they share the lowest distance), whereas PC_1 and PC_4 are the most dissimilar ones. Thus, if we had to choose only two configurations, PC_1 and PC_4 would be the most likely to cover the greatest number of t-sets according to the similarity heuristic.

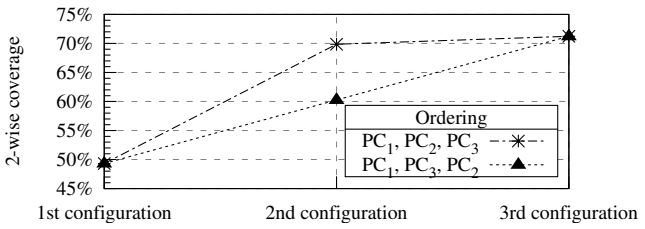


Fig. 2. A different ordering of the product configurations allows reaching faster or slower the t-wise coverage provided by these configurations.

5 Search-based Product Configuration Generation

In this section, we take benefit from the similarity heuristic to guide the generation of configurations. The objective of the generation process is to provide a set of configurations that fulfills the requirements of a test criterion. In the present context, this criterion is the t-wise coverage. If PC denotes the set of all the possible configurations and TCS a list of m configurations, this process is formally defined as:

Given: an FM, the desired number of products configurations, m, a given amount of time, t, and a function f from TCS to the real numbers, $f : PC^m \longrightarrow \mathbb{R}_+$.

Problem: finding $TCS \in PC^m$ with respect to t such as $[max(f(TCS))]$. In this context, f is the t-wise coverage achieved by the test configuration suite TCS and t is the time allowed for generating the configurations. Toward this direction, we introduce an approach, based on the (1+1) Evolutionary Algorithm [50]. Specifically, the configuration generation problem is formulated as a search-based one. The space of all the valid configurations is defined as the search space. Thus, meta-heuristic techniques can be used in order to efficiently explore this space. In view of this, similarity is used as a fitness function towards searching for configurations in this space. It enables: (a) a computationally interesting approach, independent of t and (b) prioritizing the generated configurations without necessitating much additional computation.

5.1 A Similarity-based Fitness Function

Our intuition, which will be confirmed in Section 7, is that the similarity heuristic is a relevant choice to define a fitness function f to evaluate a set of product configurations. Thus, if we consider a test configuration suite of m configurations $TCS = PC_1, ..., PC_m$, f is formally defined as follows:

$$f : \begin{array}{ccc} PC^m & \longrightarrow & \mathbb{R}_+ \\ PC_1, ..., PC_m & \longmapsto & \sum_{j>i}^{m} d(PC_i, PC_j). \end{array}$$

For instance, with reference to Sections 3 and 4, $f(PC_1, PC_3, PC_4) = d(PC_1, PC_3) + d(PC_1, PC_4) + d(PC_3, PC_4) \approx 1.53$. This function, which generalizes the similarity distances for m configurations, allows evaluating the quality of a set of configurations in terms of t-wise coverage. Indeed, the information conveyed by this function is: the higher the fitness value of the given set of m configurations, the higher the distances between the configurations, resulting in a potentially higher t-wise coverage.

Although evaluating the exact coverage would be a natural choice for a fitness function, say f_c, it would be computationally expensive for such a use. Indeed, for each configuration, it requires computing all the t-sets covered by this configuration. Consider an FM with n features and m configurations. If $\binom{n}{k}$ denotes the binomial coefficient, f_c requires to compute:

$$N = m\binom{n}{t} = \frac{mn!}{t!(n-t)!} \quad (1)$$

t-sets to evaluate the coverage of the whole set of configurations, which represents N operations. On the contrary, f requires $N' = \binom{m}{2} = \frac{m(m-1)}{2}$ distances computation plus the sum evaluation, which represents m additions.

We assume that $2 \leq t \ll n$. Therefore, the time required to compute one particular distance between two given configurations is small compared to the coverage evaluation of these two configurations, i.e., $N \gg N'$. Indeed, f does not depend on t. We also assume that one will test fewer configurations than the number of features, and thus that $m \ll n$. Especially, in a realistic and industrial context (with large FMs), the testing process is usually subjected to time and budget limitations. It thus does not allow testing as many configurations as features. It results that $N \gg N'$ and even more while t increases. Recall that we focus on t-wise, for high t-values. This fact implies a computationally lower cost for f compared to f_c. As a result, f is used as the fitness function for the configuration generation.

5.2 Search-based Approach

Classical constraint-based t-wise techniques, e.g., [18], are unable to scale to large FMs and to high values of t. This is mainly due to the number of t feature combinations. The proposed approach, which is independent of t, is composed of two steps. The first one is the generation of valid configurations using a SAT solver, and the second one is the configuration selection. The search process is formed by iteratively repeating these two steps. A similar technique that combines constraint solving and search-based approaches in a scalable way has been proposed by Harman *et al.* [51] for mutation-based test generation.

5.2.1 Generating Configurations

A SAT solver is used to produce valid configurations. Once an FM is converted into a Boolean formula [52], the solver can generate valid configurations. As a result, a search space containing only valid configurations is formed.

Typically, a product configuration is a satisfiable "model" for a given SAT solver [47]. To this end, the literals of the logical clauses (i.e., clauses represent the constraints of the FM) are assigned values. If the constraints are satisfied, one configuration is returned. However, assignments to the literals are done in a particular order which involves the following problem: no uniform exploration of the space of all the valid configurations is possible. Indeed, the order used by the solver to parse the logical clauses and literals enables their prediction. In that case, the approach always returns the same solution in a deterministic way. As a result, the configurations enumeration is driven by the order used by the solver.

To overcome this issue, and thus to get configurations in an *unpredictable* way, one solution is to randomize how the solver parses the logical clauses and the literals and how it assigns values to variables. It thus makes the solving process entirely randomized. It prevents from predicting the next configuration that will be returned. Additionally, it allows selecting configurations from the full space instead of enumerating them in a predictable order. Since it enables the use of search-based approaches in a non-biased way, the unpredictable strategy forms a contribution of this paper.

5.2.2 Selecting Configurations

The objective is to generate a test configuration suite TCS of m configurations. The proposed approach is formalized in Algorithm 1. Informally, the search-based method starts by selecting m configurations in an unpredictable way (lines 5 to 8). Then, these configurations are evaluated by the fitness function f (line 11) and prioritized (line 12). The technique used to prioritize the configurations (line 12) is presented in Section 6.2. These configurations define the initial list TCS. Then, by using the distances computed while evaluating f, the worst configuration is determined. The worst configuration is the one which has the lowest participation in the fitness function. In other words, it is the last element of TCS (line 13). The next step consists of trying to replace this configuration

Algorithm 1 Search-based Configuration Generation(m, t)

1: **input:** m, t ▷ Number of configurations to generate and execution time allowed for generating them
2: **output:** TCS ▷ Test configuration suite (prioritized)
3: $TCS \leftarrow []$
4: $S \leftarrow \emptyset$ ▷ Set of configurations
5: **for** $i \leftarrow 1$ to m **do**
6: $PC_{\text{unpredictable}} \leftarrow Request\ to\ the\ solver$ ▷ If the solver cannot give a new configuration because it has already iterated over all the valid configurations, reinitialize it
7: $S \leftarrow S \cup \{PC_{\text{unpredictable}}\}$
8: **end for**
9: $s \leftarrow size(TCS)$
10: **while** the elapsed time is lower than t **do**
11: $fitness \leftarrow f(TCS[1], ..., TCS[s])$
12: $TCS \leftarrow Global\ Max.\ Dist.\ Prioritization(S)$
13: $PC_{\text{worst}} \leftarrow TCS[s]$ ▷ PC_{worst} verifies $min\left(\sum_{k=1}^{s} d(PC_{\text{worst}}, TCS[k]\right)$
14: **repeat**
15: $PC_{\text{unpredictable}} \leftarrow Request\ to\ the\ solver$ ▷ If the solver cannot give a new configuration because it has already iterated over all the valid configurations, reinitialize it
16: **until** $PC_{\text{unpredictable}} \neq PC_{\text{worst}}$
17: $TCS.set(s, PC_{\text{unpredictable}})$ ▷ The worst configuration is replaced
18: $newFitness \leftarrow f(TCS[1], ..., TCS[s])$
19: **if** $newFitness \leq fitness$ **then**
20: $TCS.set(s, PC_{\text{worst}})$ ▷ The worst configuration is taken back
21: **end if**
22: **end while**
23: **return** TCS

by an unpredictable one got from the solver (lines 14 to 16). This replacement is conserved if and only if the fitness of the resulting list increases (lines 17 to 20). This whole process is repeated during a certain allowed amount of time t.

This technique can be considered as a genetic algorithm without crossover. It is thus an adaptation of the (1+1) Evolutionary Algorithm [50]. Indeed, instead of removing a random configuration, the worst ranked configuration, in terms of fitness, is removed.

Combining constraints with search-based methods forms a suitable approach for the configuration generation. Its use differs from both search-based [53] and similarity-based techniques [48]. Indeed, without constraint solving, generating valid configurations is almost impossible for large scale FMs. This problem is shortened by combining similarity, constraint solving and search-based approaches.

6 Product Configuration Prioritization

In this section, the similarity distances are used for prioritizing a given set of product configurations, no matter the way they have been obtained. The aim of this process is to order a set S of m configurations $S = \{PC_1, ..., PC_m\}$ according to their ability to cover t-sets. Therefore, by testing $k \leq m$ configurations, the greatest possible level of coverage, for any number of k configurations and any t value, is achieved. More formally [55]:

Given: a set of configurations, S, the set of all the permutations of S, PC_S and a function f from PC_S to the real numbers, $f : PC_S \longrightarrow \mathbb{R}_+$.

Problem: finding $S' \in PC_S$ such as $(\forall S'' \in PC_S \,|\, S'' \neq S')[f(S') \geq f(S'')]$. In this context, f is the t-wise coverage. To this end, two algorithms named *Local Maximum Distance* and *Global Maximum Distance* are introduced. They produce a list TCS, which is the result of the prioritization. They enable prioritizing efficiently the configurations with respect to t-wise.

6.1 Local Maximum Distance Prioritization

Algorithm 2 formalizes this procedure. This approach iterates over the initial unordered set of configurations S, looking for the two configurations sharing the maximum distance (line 6). These two configurations are then added to the resulting list TCS and removed from S (lines 7 to 9). This process is repeated until all the configurations from S are added to TCS.

6.2 Global Maximum Distance Prioritization

This approach is formally described in Algorithm 3. Informally, this approach selects at each step the configuration which is the most distant to all the configurations already selected during the previous steps. To this end, the two configurations belonging to S and sharing the highest distance are first added to TCS (lines 4 to 6). These two configurations are then removed from S (line 7). The next step consists in adding to TCS and removing from S the configuration sharing the maximum distance to all the configurations already added to TCS (lines 8 to 13): for each configuration of S, we sum the individual distances with the other configurations of TCS, thus giving a value for the set. Then the maximum is obtained by comparing these set values (line 10). This process is repeated until S is empty.

This technique allows having more diversity than the *Local Maximum Distance* one for $k < m$ configurations, but it is computationally more expensive. This is due to the need of calculating all the distances from one configuration to the others (Alg. 3, line 10).

Algorithm 2 Local Maximum Distance(S)

1: **input:** $S = \{PC_1, ..., PC_m\}$ ▷ Unordered set of configurations
2: **output:** TCS ▷ Prioritized test configuration suite
3: $TCS \leftarrow []$
4: **while** $\#S > 0$ **do**
5: **if** $\#S > 1$ **then**
6: Select $PC_i, PC_j \in S$ **where** $max\,(d(PC_i, PC_j))$ ▷ Take the first one in case of equality
7: $TCS.add(PC_i)$
8: $TCS.add(PC_j)$
9: $S \leftarrow S \setminus \{PC_i, PC_j\}$
10: **else** ▷ S contains only one element
11: $TCS.add(PC_i)$ **where** $PC_i \in S$
12: $S \leftarrow \emptyset$
13: **end if**
14: **end while**
15: **return** TCS

Algorithm 3 Global Maximum Distance(S)

1: **input:** $S = \{PC_1, ..., PC_m\}$ ▷ Unordered set of configurations
2: **output:** TCS ▷ Prioritized test configuration suite
3: $TCS \leftarrow []$
4: Select $PC_i, PC_j \in S$ **where** $max\left(d(PC_i, PC_j)\right)$ ▷ Take the first ones in case of equality
5: $TCS.add(PC_i)$
6: $TCS.add(PC_j)$
7: $S \leftarrow S \setminus \{PC_i, PC_j\}$
8: **while** $\#S > 0$ **do**
9: $s \leftarrow size(TCS)$
10: Select $PC_i \in S$ **where** $max\left(\sum_{j=1}^{s} d(PC_i, TCS[j]\right)$ ▷ Take the first one in case of equality
11: $TCS.add(PC_i)$
12: $S \leftarrow S \setminus \{PC_i\}$
13: **end while**
14: **return** TCS

7 Empirical Study

In this section, the configuration generation and prioritization approaches are assessed. In configuration generation, we aim at selecting configurations providing the highest coverage. In configuration prioritization, the emphasis is on maximizing the overall t-wise coverage each time a configuration is tested. The objective of this case study is to answer the four following research questions:

- [RQ1] *How does our configuration generation approach compares with state of the art tools?*
- [RQ2] *How effective is the configuration generation approach when applied on both moderate and large size feature models?*
- [RQ3] *How do our prioritization approaches compare with an interaction-based technique?*
- [RQ4] *How effective are the configuration prioritization approaches when applied on both moderate and large size feature models?*

Answering the first research question amounts to evaluating how our configuration generation approach performs compared to state of the art techniques. We expect our approach to provide a t-wise coverage close to the one achieved by the examined tools. The second research question aims at evaluating the configuration generation approach on both moderate and large FMs. Unlike existing tools, we expect our approach to scale up to $t = 6$ even on large FMs, by providing a partial but scalable t-wise coverage. We also expect it to provide higher coverage than a random technique for selecting the configurations. The third research questions amounts to evaluating how our configuration prioritization approaches perform compared to a state of the art technique based on interaction coverage. We expect our approach to provide a t-wise coverage close to the state of the art with a considerably lower execution time required as it bypasses the t-wise computation. Finally, the fourth research question aims at evaluating how our two prioritization techniques perform on both moderate and large FMs, by comparing them with a random approach. We expect our prioritization approaches to perform better than a random one.

Empirical results regarding the stated research questions are presented and analyzed. The conducted experiments[2] are performed on a Quad Core@2.40 GHz with 24GB of RAM. The study employs 114 FMs[3] divided into two categories. The first 110 FMs are small to medium size (with a number of features lower or equal to 1000); they are referred to as the *moderate* size FMs. A second subset is composed of 4 FMs of large size; they are referred to as the *large* FMs.

Regarding the moderate size FMs, 10 of them are real and 100 are artificially generated. The real FMs are taken from [24], [54] while the artificial ones are produced with the Software Product Line Online Tools (SPLOT) FM generator [54]. All involved FMs are consistent (i.e., the constraints are possible to fulfill). Details about the moderate FMs are recorded in Table 1. For each FM, it presents the number of features, the number of valid configurations[4] and the number of valid 2-sets.

2. The implemented approaches and the data used for the experiments are available at http://research.henard.net/SPL/.
3. Handled via the SPLAR [54] and the SAT solver Sat4j [47].
4. Computed via a Binary Decision Diagram.

TABLE 1
The 110 moderate size feature models involved in the empirical study. The number of features, constraints, configurations and 2-sets of the generated models are average values.

	10 Real FMs [54]										100 Generated FMs				
	Cellphone	Counter Strike Simple FM	SPL SimulES, PnP	DS Sample	Electronic Drum	Smart Home v2.2	Video Player	Model Transformation	Coche Ecologico	Printers	20 FMs	20 FMs	20 FMs	20 FMs	20 FMs
#Features	11	24	32	41	52	60	71	88	94	172	15	50	100	200	500
#Constraints	22	35	54	201	119	82	99	151	191	310	31.65	94.7	195.6	395.7	983.2
#Valid configurations (≈)	14	18,176	73,728	6,912	331,776	3.87E9	4.5E13	1.65E13	2.32E7	1.14E27	209.55	1.02E8	8.56E15	3.19E20	8.43E80
#Valid 2-sets	151	833	1,448	2,592	3,746	6,189	7,528	13,139	11,075	42,638	300.65	4,103.2	17,367.8	71,760.15	4.67E5

TABLE 2
The 4 large size feature models involved in the empirical study.

	eCos 3.0 i386pc [24]	FreeBSD kernel 8.0.0 [24]	Generated FM	Linux kernel 2.6.28.6 [24]
#Features	1,244	1,396	5,000	6,888
#Constraints	3,146	62,183	9,419	343,944
#Valid 2-sets	2,910,229	3,765,597	49,080,075	92,540,449
#Valid 3-sets (≈)	2.25E9	3.44E9	1.61E11	4.19E11
#Valid 4-sets (≈)	1.27E12	2.34E12	3.97E14	1.50E15
#Valid 5-sets (≈)	5.79E14	1.26E15	7.70E17	3.85E18
#Valid 6-sets (≈)	2.22E17	5.76E17	1.26E21	8.71E21

Regarding the large FMs, three are real, taken from [24] and one is artificially created. The details of these FMs are recorded in Table 2. It presents, for each FM, the number of features and the number of valid t-sets. The number of configurations cannot be computed in a reasonable amount of time (in days) due to the high number of constraints and features of these FMs.

For the needs of the experiment, the t-sets of the FMs for $t \geq 3$ are computed using the following procedure. First, a list of all the features of the FM is recovered. Then, all the possible t-sets are enumerated and provided to the solver to determine whether they are valid or not. For the large FMs, computing the exact number of valid t-sets is a non-trivial and time consuming task. For instance, it took around 3 days to a 10-threaded program running on our system to compute the 92,540,449 valid 2-sets of the Linux FM. As t increases, the number of valid t-sets explodes. As a result, we estimate the number of t-sets. To this end, 1,000 t-wise sets are randomly sampled and checked. Since the total number of possible t-sets of an FM is known and equal to $\binom{2n}{t}$ for n features ($2n$ because each feature is either selected or unselected), the valid t-sets can be directly estimated (law of large numbers). For example, if 800 t-sets out of 1,000 sampled are valid, the number of estimated valid t-sets is equal to $\frac{800*\binom{2n}{t}}{1{,}000}$.

7.1 Comparison with State of the Art Tools (RQ1)

In this section, we compare our approach with three state of the art tools: ACTS [56], CASA [57] and SPLCAT [18]. The latter is the most recent covering array tool available and performs for $t = 2$ and $t = 3$. The two others perform for $t = 2$ to 6.

7.1.1 Experiment Setup

We compare our generation approach with the three tools. We also consider CASA where the desired number of configurations can be specified. This approach is denoted as CASA-n. We employ the 10 real FMs of moderate size. For each FM and for $t = 2, ..., 6$, the three approaches are executed. Then, if the result is available, our approach and CASA-n are performed with the parameters corresponding to the minimum number of configurations provided by the other tehcniques. Similarly, the running time of our approach was set to the minimum one. CASA, CASA-n and our search-based approach are performed 10 times independently as they are heuristic techniques providing different solutions at each execution.

7.1.2 Experiment Results

The results of the comparison are recorded in Table 3. When the time required by an approach exceed three days (259,200 seconds), we consider its results as Not Available (N/A). Along the same lines, when none of the three tools can perform for a specific t value in less than three days, the result for this t value is not presented. Following the results of Table 3, it is clear that SPLCAT is much faster than the two other tools on almost all the FMs. In some cases, it is more than 10,000 times faster. Only CASA-n can compete in terms of time on some FMs such as the Cellphone one. As a result, our approach has been most of the time executed with the time used by SPLCAT or CASA-n.

Regarding the size of generated test configuration suites, the smaller ones are shared between CASA and SPLCAT. Our approach and CASA-n have been performed using the smallest size. Regarding the coverage achieved by our approach, one can see that it is close to 100% for all the subjected FMs.

Finally, one can observe that as the complexity of the FM increases, the tools require more time to generate the configurations and thus do not scale well to large FMs. In addition, none of the tools were able to perform for t values above 3 on most of the employed FMs, even though the complexity of these FMs is quite low.

7.1.3 RQ1 Summary

The experiments conducted for the comparison with state of the art tools bring out the following conclusions. First, **our configuration generation approach can compete with existing ones**. Indeed, our approach provides a partial t-wise coverage, with values very close to 100% for all the FMs studied. This was achieved using the minimum amount of time required among the three tools. Second, **existing tools have difficulties to scale to t values greater than 3, even on relatively small FMs.** Indeed, our experiments performed on moderate size FMs demonstrated that

TABLE 3
Comparison of the configurations' generation with ACTS, CASA and SPLCAT on the 10 smallest FMs of the empirical study for $t = 2, ..., 6$. Our search-based approach has been performed using the minimum number of configurations and minimum time performed by the other approaches, indicated in bold. N/A indicates that the generation time exceeded 3 days. The t values for which there is no result available are not represented.

		ACTS (IPOG)		CASA (avg 10 runs)		CASA-n (avg 10 runs)		SPLCAT		Search-based (avg 10 runs)		
Feature Model	t-wise	Configs.	Time	Configs.	Time	Configs.	Time	Configs.	Time	Configs.	Time	Cov.
Cellphone	2	9	2.1	**7**	0.55	7	**0.05**	8	0.15	7	0.05	98.37%
	3	13	4.4	14	1.14	14	**0.045**	**13**	0.24	13	0.045	98.12%
	4	**14**	16	14	3.79	14	**0.34**	N/A	N/A	14	0.34	99.98%
	5	14	57	14	55.15	**14**	**0.27**	N/A	N/A	14	0.27	100%
	6	**14**	399	14	6,947	14	**0.45**	N/A	N/A	14	0.45	100%
C. Strike Simple FM	2	13	3.3	8.66	1,28	**8**	0.27	10	**0.24**	8	0.24	99.34%
	3	33	70	25.33	22,16	**25**	2.21	38	**0.8**	25	0.8	99.71%
	4	94	3,278	72.67	790	**72**	**24,7**	N/A	N/A	72	24.7	98.77%
SPL SimulES, PnP	2	11	7	**9**	3.67	9	0.6	10	**0.3**	9	0.3	99.41%
	3	32	211	26.33	122.9	**26**	6.49	35	**1**	26	1	99.32%
	4	83	53,602	**72**	3,026	72	**282.5**	N/A	N/A	72	282.5	99.64%
DS Sample	2	103	506	96.33	96.16	**96**	3,13	97	**0.9**	96	0.9	98.49%
	3	N/A	N/A	**385**	12,093	385	113.2	419	**4.8**	385	4.8	99.51%
Electronic Drum	2	35	38.4	23.67	4,826	**23**	4.04	27	**0.6**	23	0.6	99.46%
	3	178	43,416	N/A	N/A	134	91.22	**134**	**2.9**	134	2.9	99.91%
Smart Home v2.2	2	17	15.5	15	28	15	3.1	**15**	**0.5**	15	0.5	99.44%
	3	75	3,731	55.67	5,182	**55**	106.7	64	**3.2**	55	3.2	99.80%
Video Player	2	15	26.5	9.33	56.4	**9**	1.5	18	**0.7**	9	0.7	99.75%
	3	46	32,687	35.67	2,542	**35**	47.02	47	**3.7**	35	3.7	99.98%
Model Transformation	2	35	187	26.33	3,165	**26**	13.6	28	**0.9**	26	0.9	99.45%
	3	N/A	N/A	N/A	N/A	130	482.7	**130**	**10**	130	10	99.91%
Coche Ecologico	2	97	2,348	**90**	156.91	90	10.6	95	**1.1**	90	1.1	99.67%
	3	N/A	N/A	N/A	N/A	378	3,694	**378**	**13**	378	13	99.87%
Printers	2	186	148,446	180.8	718.82	**180**	71.8	182	**2.8**	180	2.8	99.75%
	3	N/A	N/A	N/A	N/A	560	N/A	**560**	**139**	560	139	99.81%

for 7 FMs out of 10, none of the three tools was able to provide results within 3 days from $t = 4$. The 3 FMs on which the tools worked are the smallest one, and for two of them, results are only available up to $t = 4$. Overall, the fastest tool among the three is SPLCAT.

7.2 Product Configuration Generation Assessment (RQ2)

Here, we assess the ability of the proposed approaches to cover t-sets. To this end, we evaluate our approach on all the moderate size FMs for 2-wise and compare it to SPLCAT. We limit to 2-wise since SPLCAT does not scale well to 3-wise or above (at least in a reasonable amount of time, in days) for the subjected FMs. As far as we know, our *search-based* approach is the only one which allows scaling to any t value, even for large FMs. Finally, since no other technique can serve as a basis for comparison for the large FMs, we compare the search-based approach with configurations selected in an unpredictable way from the SAT solver (Section 5.2.1). In the following, this approach will be referred to as the *unpredictable* one and will also serve as a comparison basis.

7.2.1 Moderate Feature Models

Here, we compare the search-based approach with both the unpredictable approach and SPLCAT. This study is only based on the 2-wise coverage and considers only the moderate FMs.

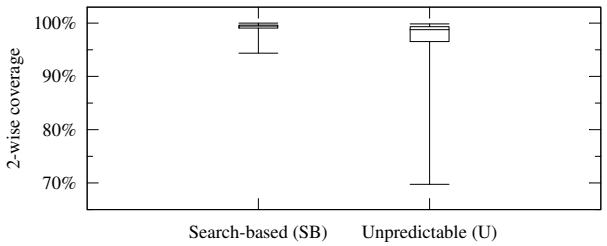
Experiment Setup: To enable a fair comparison, the search-based and unpredictable approaches generate sets of configurations of the same size as those provided by SPLCAT. The search-based approach is allowed to run for one minute and is performed 10 times per FM. For the 100 generated FMs, the results presented are averaged on all the FM.

Experiment Results: The results are presented in Figure 3. SPLCAT is not represented as it always achieves 100% of coverage. The results for the generated FMs are depicted by Figure 3a and the results for the real FMs are represented on Figure 3b. It appears that the proposed search-based approach, as an approximation technique, is close to SPLCAT. Indeed, in the best case, it is able to achieve 100% of 2-wise coverage with only 1 minute of processing time allowed. In the worst case, 95% is achieved on both the generated and real FMs. In addition, the search-

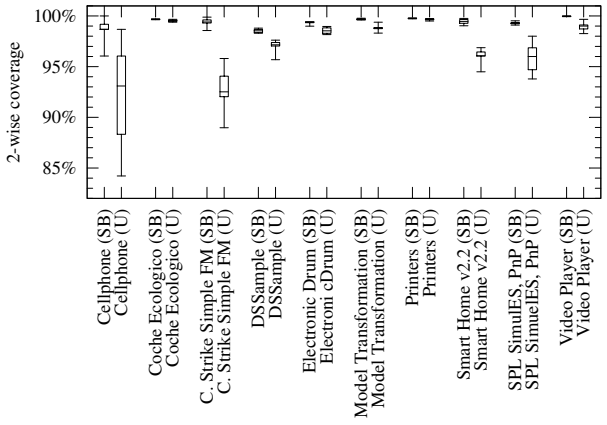
based approach is much more stable than the unpredictable one, which can drop down to 69% of coverage in the worst case. Although 100% of coverage might be desirable, the focus of our approach, as explicitly stated in the introduction section, is the partial but scalable t-wise coverage.

Besides, the performance of SPLCAT varies. For FMs up to 200 features, SPLCAT requires less than a minute. However, it takes around 6.2 minutes for the FMs of more than 200 features, and around 159 minutes for the 1,000 features ones.

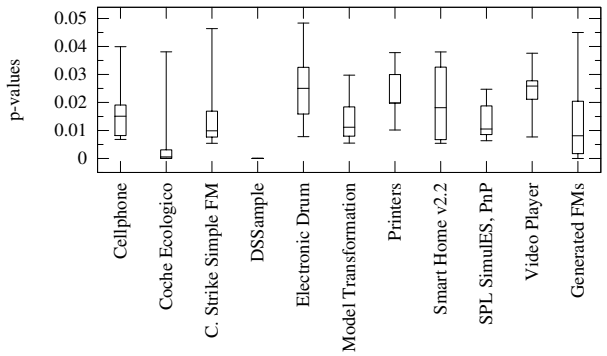
Finally, to evaluate whether the difference between the search-based approach and the unpredictable one is statistically significant, we followed the guidelines suggested by Arcuri and Briand [58]. To this end, we performed a Mann-Whitney U Test. It is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other. We obtain from this test a probability called p-value which represents the probability that the two samples are equal. It is conventional in statistics to consider that the difference is not significant if the p-value is higher than the 5% level. For each run per FM, we computed the p-value between the two approaches. It results in 10 p-values for each FM. Figure 3c depicts the distribution of the p-values for the real and generated FMs. For the generated FMs, the p-values are represented all together. Since all the p-values are below the significance level 5%, the difference between the two approaches is considered as statistically different.



(a) Generated FMs (averaged)



(b) Real FMs



(c) Result of the Mann-Whitney U Test between the search-based and the unpredictable approach on the moderate size FMs (equal hypothesis)

Fig. 3. Product configuration generation on the 110 moderate FMs for $t = 2$ (1 minute execution for the search-based approach, 10 runs for each approach. The execution time of the unpredictable approach is few seconds.).

7.2.2 Large Feature Models

Scaling to large FMs is a quite difficult task, even for 2-wise. Neither SPLCAT nor the tools we studied (see Section 7.1) are able to scale well to these sizes [9]. On the contrary, the search-based approach efficiently deals with the t-wise combinations where no other approach is able to do so, by producing a partial coverage. Here, we evaluate the t-wise coverage ability of the search-based and unpredictable approaches on the large FMs for $t = 2, ..., 6$.

Experiment Setup: To estimate the t-wise coverage of the product configurations, we use a process similar as the one used to calculate the t-sets of an FM and described in the beginning of Section 7. The sampling process is repeated 10 times per each examined t value ($t = 2$ to $t = 6$) with samples of size 100,000. The search-based and unpredictable approaches are executed on all the large FMs to produce 5 times 50 and 100 configurations, with the time restriction of 30 minutes. Another experiment involves the generation of 1,000 configurations and the recording of the coverage over the runs of the search-based approach.

Experiment Results: The results are recorded in Table 4. This table presents the mean coverage achieved with respect to t-wise per FM and per approach. Additionally, it records the standard deviation of these values. A score above 95% with respect to 2-wise is achieved by both the approaches and for all the studied FMs when producing 50 configurations. With respect to 6-wise, scores of 35% to 50% are achieved. By producing 100 configurations, higher scores are achieved for both the approaches. It should be mentioned, based on the standard deviation values recorded in Table 4, that a small variation on the achieved coverage is observed. It is a fact indicating that the approaches are quite stable.

Generally, the search-based strategy provides a higher coverage compared to the unpredictable ap-

TABLE 4
T-wise coverage (%) for the large FMs with 50 and 100 configurations. The search-based approach was allowed to run for 30 minutes. The execution time required by the unpredictable approach is few seconds.

		Search-Based		Unpredictable		Search-Based		Unpredictable	
		50 product configurations				100 product configurations			
FM	t-wise	Mean	Std.Dev.	Mean	Std.Dev.	Mean	Std.Dev.	Mean	Std.Dev.
eCos	2	99.04	0.10	98.12	0.34	99.60	0.07	99.43	0.17
	3	94.38	0.26	92.28	0.52	97.55	0.14	96.89	0.36
	4	83.40	0.44	80.92	0.53	91.44	0.23	90.41	0.49
	5	67.26	0.49	65.59	0.50	80.02	0.29	79.29	0.51
	6	49.98	0.45	49.43	0.45	64.96	0.29	64.90	0.48
FreeBSD	2	98.89	0.11	97.95	0.27	99.31	0.10	99.17	0.16
	3	95.67	0.16	92.77	0.44	97.89	0.12	96.70	0.30
	4	86.56	0.24	81.88	0.53	93.63	0.20	90.65	0.40
	5	69.97	0.26	65.23	0.48	83.68	0.28	79.29	0.40
	6	50.12	0.22	46.69	0.37	67.57	0.29	63.22	0.36
5000f. gen	2	95.92	0.12	94.93	0.29	98.30	0.08	97.83	0.19
	3	85.94	0.20	84.04	0.33	92.31	0.16	91.19	0.25
	4	70.50	0.21	68.24	0.30	80.86	0.25	79.22	0.25
	5	52.89	0.18	50.87	0.25	65.39	0.25	63.65	0.25
	6	36.63	0.18	35.18	0.23	48.92	0.25	47.44	0.23
Linux	2	97.74	0.16	97.03	0.23	98.74	0.09	98.46	0.15
	3	93.03	0.21	91.86	0.28	96.03	0.13	95.47	0.21
	4	82.67	0.24	81.25	0.27	90.28	0.18	89.48	0.22
	5	65.77	0.23	64.50	0.25	79.33	0.20	78.40	0.21
	6	46.48	0.18	45.63	0.20	63.08	0.21	62.25	0.23

proach, especially for high values of t. This is true for all the t-wise coverage measures. Allowing more time to the search-based technique should increase the gap with the unpredictable approach since the iterations improve the set of configurations. However, the results are based on the selection of 50 and 100 configurations. Therefore, the maximum difference between the two approaches lies between the coverage of the unpredictable selection and the maximum possible coverage achievable with 50 or 100 configurations. Achieving 100% of t-wise coverage with 50 or 100 configurations seems to be impossible for the large FMs. It is expected that more configurations are needed to achieve 100% of coverage.

To evaluate whether the difference between the two approaches is statistically significant, we perform a Mann-Whitney U Test a the same lines as explained in Section 7.2.1. To this end, we applied the following procedure. For each t-value, each number of configurations (50 and 100) each of the 30 runs and each of the 10 t-sets sample, we evaluated the p-value resulting from the test between the search based approach and the unpredictable one. It results in $5 \times 2 \times 30 \times 10$ p-values per FM. Figure 4 presents the distribution of these 3000 p-values per FM. The resulting p-values are below the level of significance of 5%, fact indicating that the two approaches are significantly different.

Table 5 records the coverage achieved by the search-based approach each 5,000 runs repetitions for 1,000 configurations with respect to 6-wise. Here, we observe that a higher level of coverage is achieved with more configurations. For instance, the search-based approach achieves 90,671% of 6-wise coverage for the Linux FM. It also shows, as it can be expected for a search-based approach, that allowing more processing time to the approach allows reaching a higher coverage. Indeed, at each 5,000 runs, the coverage recorded is higher than the previous one. Here, the unpredictable approach, represented by the "0 run", is also the initialization stage of the search-based strategy (Alg. 3, lines 5 to 10). For example, considering the eCos FM, 94.191% of 6-wise coverage is achieved at the initialization. After 15,000 runs, it is 95.343%, which represents $\approx$ 2.475744E15 additional 6-sets covered compared to the unpredictable approach. The number of valid t-sets is extremely high (see Table 2) and thus, a small increase in the coverage represents a high increase in the number of additional valid t-sets covered. Finally, the 15,000 runs require about 10 to 20 hours of processing time per FM.

7.2.3 Fitness Function

So far, the presented results suggest that the search-based approach is effective and able to scale to large

TABLE 5
6-wise coverage and fitness evolution over time for the search-based approach on the large FMs with 1,000 product configurations.

	0 run (=unpred.)		5,000 runs		10,000 runs		15,000 runs	
	Coverage	Fitness	Coverage	Fitness	Coverage	Fitness	Coverage	Fitness
eCos	94.191%	271,880	94.225%	286,304	94.263%	288,039	95.343%	288,818
FreeBSD	76.236%	294,184	76.395%	299,962	76.465%	300,892	76.494%	301,634
Generated FM	82.986	258,763	84.492%	263,243	84.605%	263,974	84.778%	264,362
Linux	89.411%	296,661	90.404%	298,709	90.640%	299,114	90,671%	299,363

FMs. Scalability is reached thanks to the ability of the similarity fitness function to mimic the t-wise coverage. To illustrate this fact, Table 5 records the fitness function values with respect to 6-wise coverage for the large FMs as the search-based approach evolves. It shows that the fitness increases with the coverage over the runs of the approach. The same trend holds for all the FMs and values of t considered in this study. Figure 5 illustrates the correlation between the fitness and the t-wise coverage for the Linux FM. Therefore, the assessment of a set of configurations can be performed without computing any t-set, thanks to the fitness function. Recall that computing the t-sets requires vast computational resources (Section 5.1, Eq. 1).

7.2.4 RQ2 Summary

The configuration generation experiments emphasize the following outcomes. First, **the similarity heuristic and the fitness function driving the approach form an efficient guide toward the configurations selection.** The search-based product configuration generation mimics the t-wise coverage, does not depend at all on t and thus, it avoids the combinatorial explosion due to the combinations of t features. Second, **the proposed technique is the first one, to the authors' knowledge, which scales well to large FMs while achieving a decent level of t-wise coverage** (depending on the number of configurations desired). Finally, in addition to be a close approximation of SPLCAT, it is more flexible than the latter as it allows specifying the processing time and the number of desired configurations. These are characteristics conforming to an industrial context where the testing process is subjected to budget constraints.

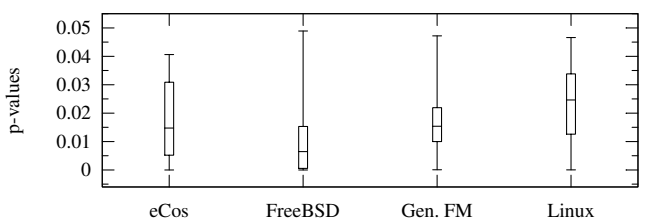


Fig. 4. Result of the Mann-Whitney U Test between the search-based and the unpredictable approach for $t = 2...6$ on the four large FMs (equal hypothesis).

7.3 Similarity-based Product Configuration Prioritization Assessment (RQ3 & RQ4)

This part evaluates the proposed prioritization approaches. To this end, we compare them with a prioritization technique based on interaction coverage from Bryce and Memon [37]. This technique is commonly used in CIT studies and it is based on t-wise interaction coverage. Since similarity forms an alternative to the t-wise evaluation, it is natural to compare with it. In the remainder of this paper, we refer to this approach as Interaction-based.

The first experiment focuses on $t = 2$ for the moderate FMs, due to the limitations of SPLCAT (see Section 7.2 and 7.2.1). The second experiment demonstrates the ability of the similarity-based approaches to scale to any t value for the large FMs. This second part does not consider SPLCAT and the Interaction-based approach [37] given their inability to scale, as demonstrated by our results (see Section 7.3.1).

To compare the prioritization approaches, the area under curve is evaluated. According to Do and Rothermel [59], "the area under the curve represents the weighted average of the percentage of faults detected over the life of the test suite. This area is the prioritized test suites average percentage faults

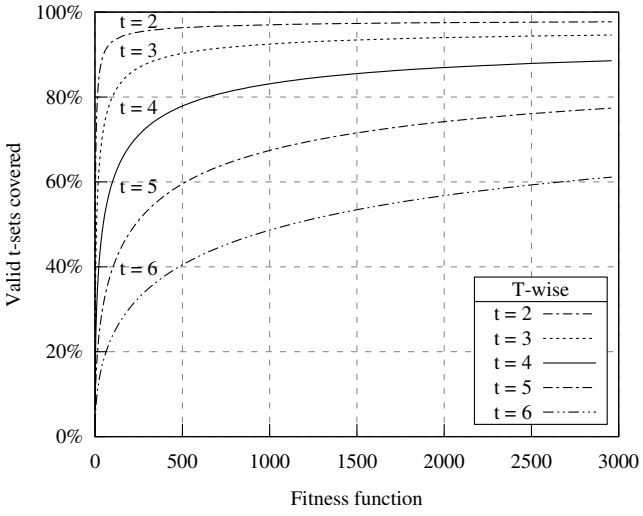


Fig. 5. Fitness function correlation with t-wise coverage for the Linux kernel feature model.

TABLE 6
Prioritization results: area under curve (scale 1:1,000).

	Case I	Case II	Case III	Case IV / 100 confs.					Case IV / 500 confs.					Case V / 100 confs.					Case V / 500 confs.				
Technique \ T-wise	2	2	2	2	3	4	5	6	2	3	4	5	6	2	3	4	5	6	2	3	4	5	6
Random	7.99	7.74	8.93	9.23	8.34	7.10	5.57	4.05	49.06	47.69	45.09	40.88	35.22	8.65	7.33	5.67	4.02	2.67	47.47	44.70	40.42	34.40	27.40
Local Max. Dist.	8.33	7.89	9.07	9.28	8.43	7.17	5.61	4.07	49.16	47.77	45.15	40.97	35.32	8.89	7.68	6.13	4.45	3.03	47.76	45.28	41.44	36.14	29.55
Global Max. Dist.	8.43	8.02	9.19	9.33	8.48	7.22	5.65	4.11	49.23	47.92	45.46	41.32	35.66	9.06	8.00	6.56	4.91	3.37	48.15	46.19	42.95	38.03	31.71
Interaction-based	8.43	8.03	9.20																				
SPLCAT	8.37																						

detected metric (APFD)." In our study, we measure the interaction coverage instead of the percentage of faults. Hence, the area under curve represents the effectiveness of the studied approaches. This area is the numerical approximation of the integral of the discrete coverage curve and is computed using the trapezoidal rule, i.e., $\int_a^b g(x)dx \approx (b-a)\frac{g(a)+g(b)}{2}$. Thus, for each prioritization method, if $cov(x)$ denotes the percentage of t-wise coverage achieved with the x-th configuration, then the area value is given by $\sum_{i=1}^{99} \frac{cov(i)+cov(i+1)}{2}$. A higher area under curve value expresses a more effective prioritization.

7.3.1 Moderate Feature Models

This part of the experiments compares our prioritization techniques to SPLCAT and the Interaction-based approach for $t = 2$. SPLCAT does not provide an independent prioritization approach as we do but it tries to cover the maximum of 2-sets each time a configuration is added, effectively implementing the greedy heuristic for prioritization [55]. The resulting configurations can thus be considered as ordered for covering faster the highest amount of 2-sets.

Experiment Setup: For each moderate FMs, three different sets of configurations are used to apply the prioritization techniques. The first set is the set of configurations produced by SPLCAT (Case I). The second one is a set of n configurations, where $n = \frac{\#features}{2}$, selected with the unpredictable method (Case II). Finally, the last set is composed of the configurations generated by SPLCAT plus the same amount of configurations selected by the unpredictable method (Case III). Using these different sets allows ensuring that the prioritization approaches are relevant whatever the nature of the configurations.

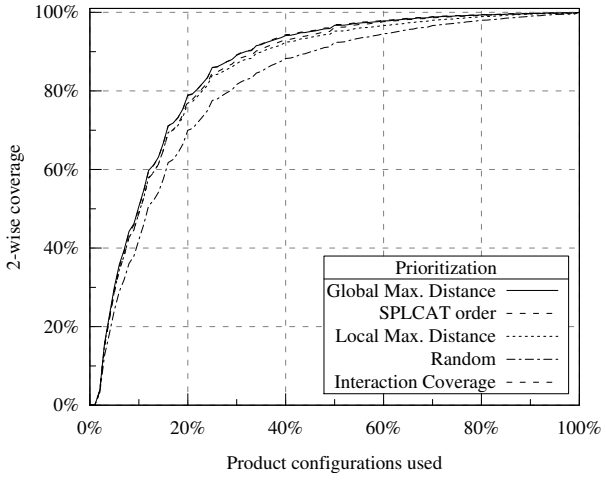
All these sets of configurations are randomized before executing the prioritization techniques. This practice ensures that our approaches are independent of the original order. On each of the three cases and for each FM, a random prioritization is averaged 10 times. Cases II and III are independently repeated 10 times to avoid any bias from the initial set of configurations. For each approach and each independent repetition, the execution time is recorded.

Experiment Results: Table 6 presents the area under curve for each case and technique. Recall that a higher surface value indicates a better prioritization. With respect to Table 6 and focusing on Case I, we observe the following ordering: Random $<$ Local Maximum Distance $<$ SPLCAT $<$ Global Maximum Distance $\approx$ Interaction-based. For Case II and Case III, the order Random $<$ Local Maximum Distance $<$ Global Maximum Distance $<$ Interaction-based is observed. However, the Global Maximum Distance approach performs almost equally as the Interaction-based one (difference of 0.01 in the area under curve). It shows the ability of the similarity heuristic to mimic the t-wise coverage. In addition, it also performs better than the Random and Local Maximum Distance.

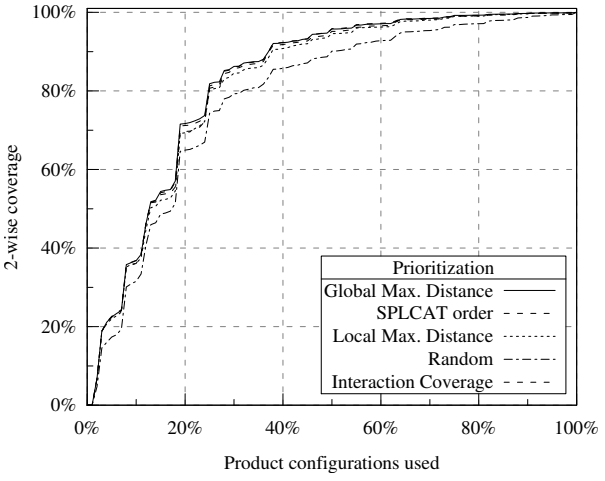
Figure 6 illustrates this behavior. For each case and category of FM (real and generated), the results are averaged on all the FMs of the category by normalizing the number of configurations selected from 0 to 100%. For instance, with respect to Case I and the generated FMs (Figure 6a), the Global Maximum Distance prioritization approach enables covering more than 90% of the 2-set with only 30% of the configurations. On the contrary, the random prioritization needs around 50% of the configurations. For Case II and Case III, the same trends are observed. These results emphasize that the prioritization techniques are either able to perform similarly (Local Maximum Distance) or better (Global Maximum Distance) as both the SPLCAT and the interaction-based approach.

Regarding the execution time, consider Figure 7. It shows the average execution time for the considered prioritization approaches on the real FMs (Figure 7a) and on the generated FMs (Figure 7b). SPLCAT is not considered as it is a configuration generation approach. From these figures, it is clear that the Interaction-based technique has difficulties to scale. This is due to the expensive computation of the t-wise interactions (see Section 5.1, Equation 1). For instance, consider the generated FMs (Figure 7a). For models of 200 features, it requires around 50,000 milliseconds. For FMs of 500 features, the execution time increases to more than 10^6 milliseconds.

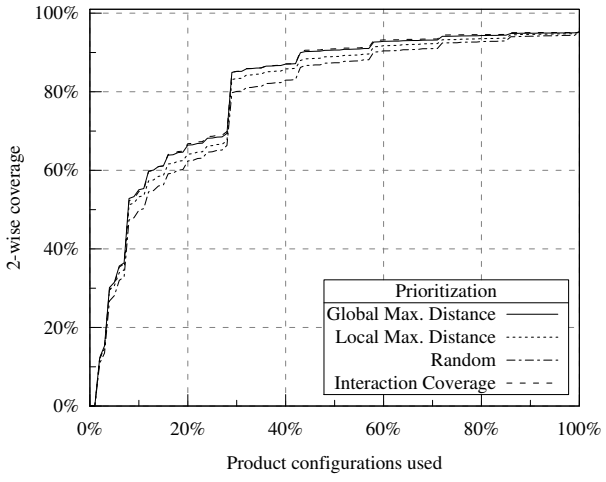
On the contrary, the Local and Global Maximum Distance approaches are significantly less impacted by the complexity of the FM. Finally, Table 7 shows



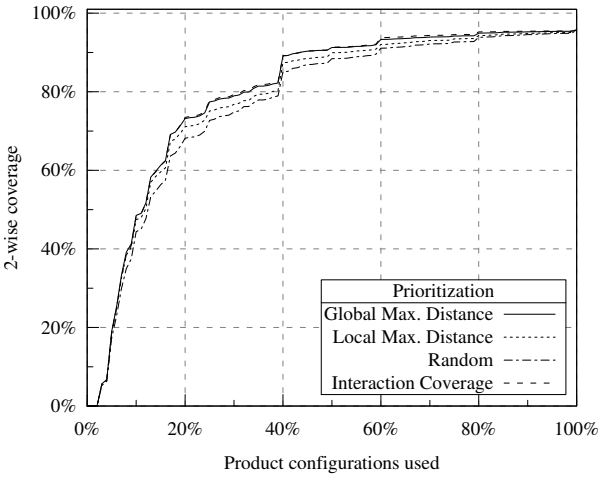
(a) Case I: SPLCAT configs / Generated FMs



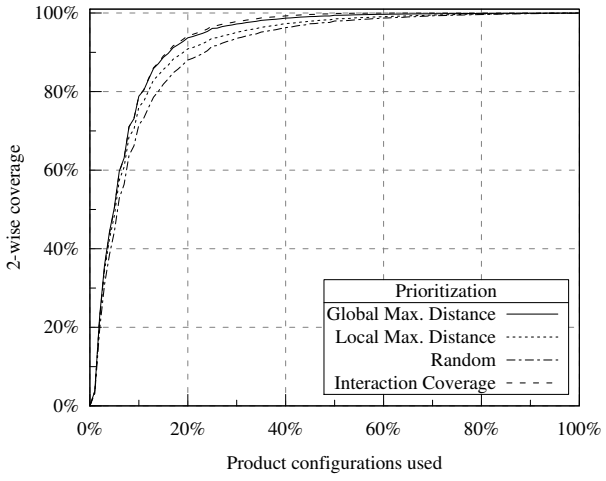
(b) Case I: SPLCAT configs. / Real FMs



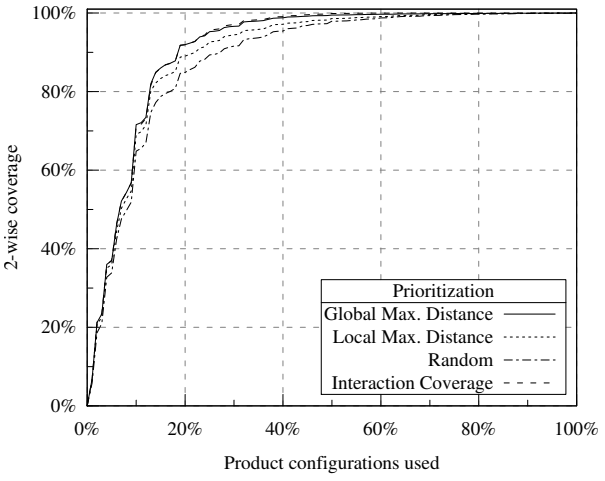
(c) Case II: unpredictable configs. / Generated FMs



(d) Case II: unpredictable configs. / Real FMs



(e) Case III: SPLCAT + unpredictable configs. / Generated FMs



(f) Case III: SPLCAT + unpredictable configs. / Real FMs

Fig. 6. Prioritization on the moderate size FMs ($t = 2$). Each approach has been performed 10 times per FM.

the execution time for the different cases. Overall, the Global Maximum Distance approach provides a little overhead compare to the Local Maximum Distance. Given the fact that the Global Maximum Distance performs similarly to the Interaction-based technique with a considerably lower computational overhead, its use is advisable.

TABLE 7
Prioritization results: execution time in milliseconds.

	Case I	Case II	Case III	Case IV / 100 confs.	Case IV / 500 confs.	Case V / 100 confs.	Case V / 500 confs.
Local Max. Dist.	241.3	263.4	288.1	1,693	45,437	1,602	46,299
Global Max. Dist.	250.1	271.2	299.4	1,764	47,310	1,698	48,267
Interaction-based	45,256	61,371	111,864				

7.3.2 Large Feature Models

This part of the study assesses the Global Maximum and Local Maximum Distance prioritizations on the large FMs for $t = 2, ..., 6$. The Interaction-based approach is not considered given its difficulty to handle moderate size FMs, as shown in the previous section.

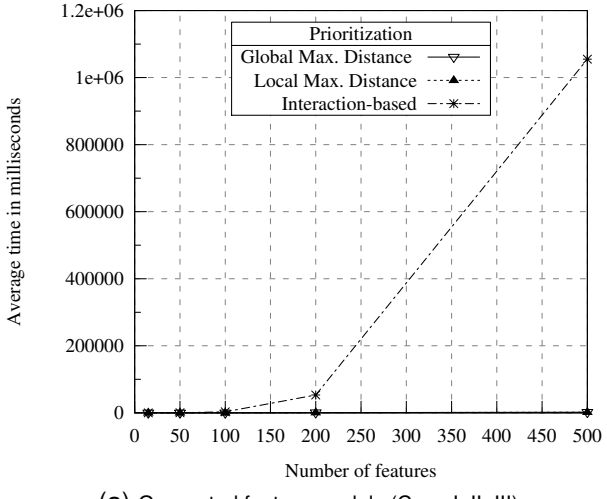
Experiment Setup: We generate two sets of 100 and 500 configurations containing dissimilar configurations (Case IV) and two sets of the same sizes containing half similar and dissimilar configurations (Case V). We choose these two kinds of sets of configurations since the prioritization approaches are similarity-driven and can thus be influenced by the nature of the used sets. Indeed, applying these approaches on sets containing dissimilar configurations can be less effective than applying them on sets containing similar configurations. We randomize each set of configurations and execute the Local Maximum Distance and Global Maximum Distance prioritizations on each of them. We also produce 10 random orderings to compare with our approaches. This practice shows that the prioritization techniques are not affected by random orders.

Experiment Results: As for the results presented in Section 7.3.1, we evaluate the area under curve. The results are recorded in Table 6, Cases IV and V. Random is averaged on 10 runs for each value of t. The presented values are averaged on the 4 large FMs. We observe the following ordering for both Case IV and Case V: Random $<$ Local Maximum Distance $<$ Global Maximum Distance. Thus, the prioritizations approaches are relevant for finding the dissimilarities in the sets containing both similar and dissimilar configurations. The Global Maximum Distance prioritization tends to be the most relevant approach.

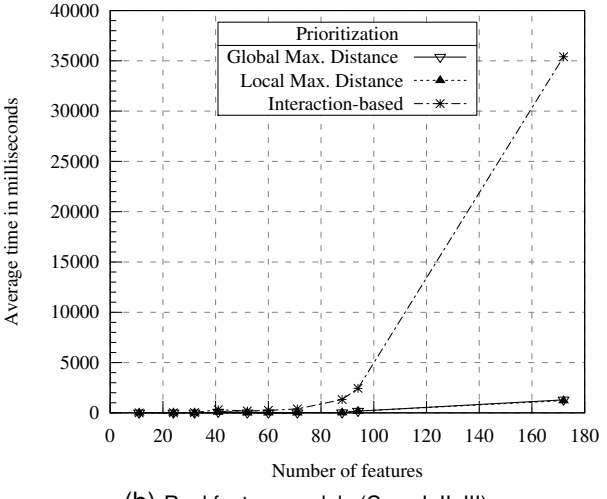
As expected, when configurations are already dissimilar (Case IV), the gain is lesser than when the set of configurations is any (Case V). Additionally, Figure 8 presents the t-wise coverage difference between the Global Maximum Distance prioritization and the random ordering for 500 configurations, averaged on the 4 FMs. For Case IV (Figure 8a) and $t = 4$, 3% of difference is observed with 30 configurations selected. For Case V (Figure 8b), 14% of difference is observed with 100 configurations for $t = 6$.

7.3.3 RQ3 & RQ4 Summary

The experiments conducted for the prioritization bring out the following conclusions. First, the Global Maximum Distance performs similarly to the Interaction-based approach. However, the **Global Maximum Distance approach is significantly faster and less sensitive to the complexity of the FM than the Interaction-based one**. Thus, it forms a scalable approach for prioritizing product configurations. Second, **the most relevant configurations contributing**



(a) Generated feature models (Case I, II, III)



(b) Real feature models (Case I, II, III)

Fig. 7. Execution time for the prioritization on the moderate size feature models ($t = 2$). Each approach has been performed 10 times per feature model.

to t-wise coverage are the most dissimilar ones. This is enabled by the similarity heuristic. Finally, **the proposed prioritization approaches are able to prioritize any set of configurations**, by looking for the dissimilarities. This is performed without computing any t-sets and regardless of the value of t.

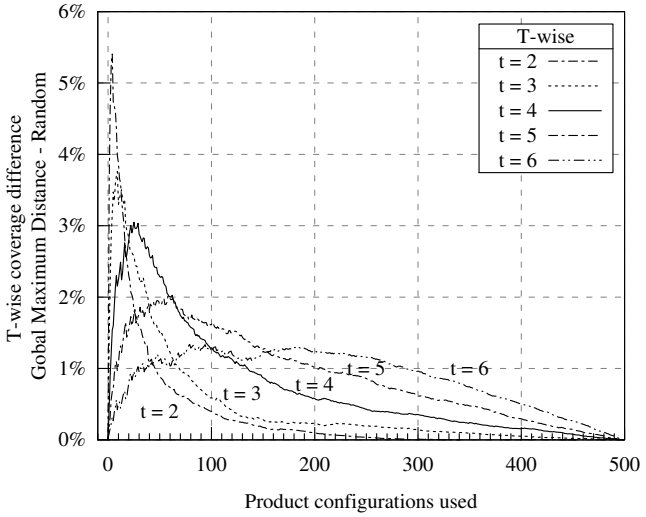
8 Discussion

This section first discusses the interaction fault detection ability of test configuration suites. Then, it presents practical implications and further applications of our approaches. Finally, the limitations of our techniques and the threats to the validity of the conducted experiments are highlighted.

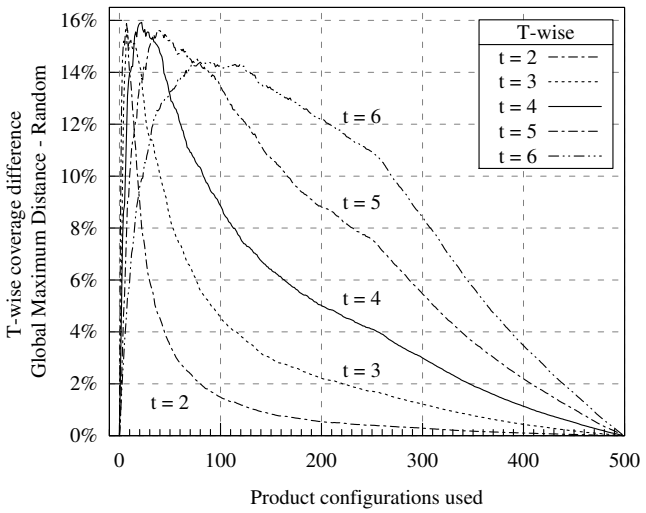
8.1 Detecting T-wise Interaction Faults

Failure due to interactions are difficult to detect as they occur when several features are involved together. Generally, each feature can be tested independently, e.g., using unit testing. Highlighting t-wise faults is more difficult. Arcuri and Briand [11] established the lower bound for the probability of a random test suite to trigger at least one failure related to t-wise. However, this bound is only valid in the context of CIT without constraints.

In our context, features are constrained and the above-mentioned results cannot be applied directly. Providing theoretical results, such as those of Arcuri and Briand [11] is not possible in the presence of constraints. This is due to the fact that constraints are specific to each FM. Therefore, in our work, we turn to empirical analysis. In order to complete the experiments with reasonable resources, we restrict the experiments to t-wise interaction faults for $t = 2$ to 6. If we consider that all the t-wise interactions of the SPL have the same probability to trigger a fault and that concrete test cases derived from a selected product configuration expose all the feature interaction faults that are present in this configuration, then the probability that a fault is found by a test configuration suite can be represented by the t-wise coverage [11]. This probability is calculated by summing the t-wise coverage for all the examined t values ($t = 2, ..., 6$).



(a) Case IV: Dissimilar set of configurations



(b) Case V: Similar + dissimilar configurations

Fig. 8. Global Maximum Distance VS Random prioritization on the four large feature models.

Figure 9 depicts the probability of finding faults for the large FMs for all the t-values in the context of prioritization. This probability is obtained by summing the probabilities for all the FMs. Compared to a random ordering, a difference of about 15% in the probability to find a fault can be observed with around 100 product configurations. It means that with the first 100 configurations proposed by our approach, we reach a probability of finding a fault equal to 57% whereas a random ordering would need more than 200 configurations to reach the same probability.

Regarding the configuration generation, Table 8 presents the estimation of the fault detection rate achieved by the search-based and the unpredictable approaches on the 4 large FMs, for $t = 2, ..., 6$. For 50 configurations, the search-based approach yields an estimated interaction fault detection rate of 76.86% and the unpredictable one a rate of 74.63%. Thus, an average difference of more than 2% is observed between the two techniques with only 50 configurations.

8.2 Practical Implications

While using existing tools such as SPLCAT, we realized that existing approaches which implement a covering array technique already perform a prioritization with respect to t-wise interactions. This is due to the construction of these techniques which try to cover the maximum amount of t-sets with each new configuration. As a result, the outcome of our prioritization techniques on small or moderate size FMs is limited, as existing approaches already perform a prioritization. However, even in this case,

our techniques perform better than the existing ones. The benefit of our techniques is more evident in the context of large SPLs, where other tools cannot work. In addition, our prioritization methods can operate on any set of configurations. They can therefore be used in combination with existing approaches and tools.

Our generation approach requires an FM. It can work on any FM, regardless of its complexity. As a result, the system abstracted by the FM has no impact on our approach. When the available model is not an FM, and if the model can be translated to a boolean one, then our approach can also be applied. Indeed, it is still possible in that case to employ transformation rules, such as [28] in order to transform the model to a boolean one. If such a transformation is not available, our technique can be combined with a Satisfiability Modulo Theory (SMT) solver to handle non-boolean models. Finally, our technique does not rely on code or existing test cases. Such artifacts may not be available at an early stage of the system development or hard to analyze directly due to their size and complexity. As a result, such cases are well suited for applying our approach.

The number of product configurations to generate and the amount of time allowed to generate them are parameters of our search-based approach. These parameters aims at making the testing process more flexible. Indeed, existing approaches [9] generate all the configurations necessary to cover all the t-sets. The problem is that they may take a large amount of time to perform this full coverage and they may generate too many configurations. To the authors' knowledge, our approach bestows a unique feature which aims at maximizing the t-wise coverage for the specified amount of configurations, given the specified amount of time. As a result, it gives a partial coverage but also makes the testing process more practical for large SPLs. In any cases, for large FMs, it is not possible to evaluate all the t-wise interactions.

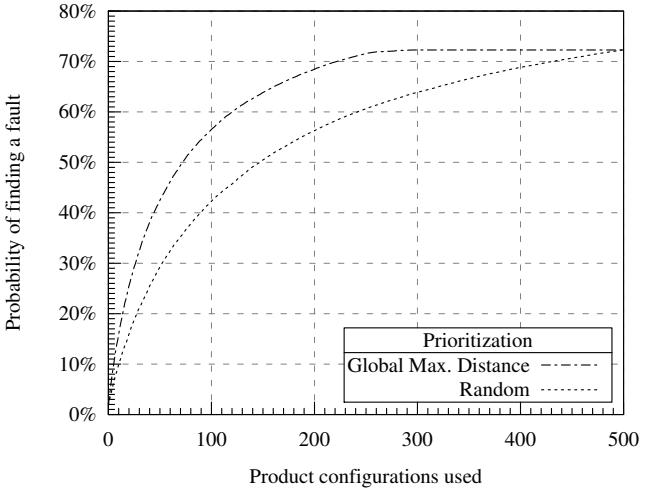


Fig. 9. Estimation of the interaction fault detection rate of the Global Maximum Distance and Random prioritizations for all the t-values. It uses the configurations resulting from the prioritization experiment on the four large feature models (Case V).

TABLE 8
Estimation of the interaction fault detection rate on the 4 large feature models for $t = 2, ..., 6$ for the search-based (SB) and unpredictable (U) approaches

	50 confs.		100 confs.	
	SB	U	SB	U
eCos	78.81%	77.28%	86.70%	86.19%
FreeBSD	80.24%	76.90%	88.41%	85.80%
5000f. gen.	71.26%	68.31%	71.75%	71.26%
Linux	77.14%	76.05%	85.49%	84.81%
avg	76.8625%	74.635%	83.0875%	82.015%

Besides, our approach scales to large FMs. While using constraint solvers, we observed that solving constraints is a time consuming task and thus an obstacle to scalability. In addition, calculating the t-wise coverage is difficult for large FMs since all the t-sets of the configurations have to be considered. Our generation approach uses a SAT solver only for generating configurations satisfying the constraints of the FM. The prioritization and generation techniques are driven by a similarity heuristic which mimics t-wise coverage and does not require to compute any combination of feature. As a consequence, one strength of the proposed approach is that it maximizes the coverage for any t value. On the contrary, existing approaches focus only on a given t-value. This results in maximizing the fault detection up to t while leaving aside the higher strengths of t [11]. Thus, our approach has a clear advantage over existing ones.

8.3 Further Applications

The propositions made in this paper have potential application to other possible issues related to CIT or SPL testing. For example, considering the evolution of an SPL over time [44], our approach can also be relevant. In that context, different versions of the FM exist (the original and the evolved FMs). Each version represent a model of the SPL. Therefore, taking into account the evolution over time implies modifying the way the distances are computed. The aim is to focus on the features that have changed or that have been added to the evolved version of the SPL. Thus, by defining a distance measure which ignores the unchanged features from the calculation, the proposed approach is generate and prioritize configurations over the interactions of the modified or new features.

Along the same lines, other approaches [19], [60], [61] attribute different importance to t-wise combinations. This practice can reduce the complexity of the problem since only the most important combinations are considered. However, our approach also targets

on early development stage where no code or system is available. Furthermore, even in the case where the system code is available, these approaches face the usual pitfalls of dynamic analysis. Thus, scalability issues, problems of handling system calls or memory constructs restrict the application of such approaches on large scale systems. In addition, our approach is somehow orthogonal to these techniques due to the generation and prioritization. In order to take into account the relative importance of feature combinations, there is a need to assign weights representing the importance of the interactions. These weights can be assigned based on the use of dynamic approaches. Our approach can handle weight by defining an appropriate distance measure. In this work, we consider all the features and features combination as equal. The importance of features or combination of features is a worth studying problem which has been left open for further research, adding one objective to be considered by the prioritization algorithm [62].

8.4 Limitations

The first limitation of our configuration generation approach is that it does not provide a full coverage. Indeed, we perform a partial but scalable t-wise coverage. In other words, we try to maximize the t-wise coverage achieved for a given number of product configurations, within the specified amount of time provided. This is not a problem as evaluating all the t-wise interaction is difficult for very large SPLs. In addition, in an industrial context, the testing budget is typically limited, preventing all the products of the SPL from being tested. Thus, a smaller to 100% coverage is going to be achieved. The second limitation is that our approach works only with FMs. It may also work with other models that can be translated to boolean ones. Alternatively, an SMT or a CSP solver could be used to satisfy non-boolean constraints. Finally, we depend on the scalability of constraint solvers and the overhead they induce. We believe that this is not important as we did not find any concrete FM raising such an issue. We acknowledge this as a potential limitation for the cases of FMs significantly larger than the studied ones.

8.5 Threats to Validity

Although we used various FMs, there is an *external validity* threat. Indeed, we cannot ensure that the proposed strategies will provide similar results on different sets of FMs (larger or more constrained). To reduce this threat, we used a relatively large set of 114 FMs of different sizes, combined real and generated FMs to cope with a variety of situations. Additionally, potential errors in our implementation could affect the presented results and lead to *internal validity* threats. To diminish these threats, we divided the implementation into sub stages to have a better control on each of the steps composing the proposed approaches. The comparison with existing tools also gave us confidence in our implementation. In addition, in order to enable reproducibility and to reduce the abovementioned threats, we made our implementation and the experiment data publicly available. Besides that, to prevent as possible a *construct validity* threat, we sampled each technique on 10 runs.

Another threat concerns the identification of faulty interactions by the actual testing of a concrete product. It is assumed that testing a software product ensures revealing the interaction faults that it contains. While this holds, it is a common assumption made by all the CIT approaches. CIT techniques require to cover at least once each t-wise interaction, supposing that executing the test configuration suite will effectively reveal the faulty interactions. Additionally, this assumption is in line with the structural testing (code coverage) approaches. Branch coverage forms a testing requirement in many software standards, e.g., [63]. However, covering branches or statements assumes that executing parts of the code will trigger the faults that they contain [64]. Besides, it should be clear that to reveal a faulty interaction, a test should exercise this interaction. Suppose that a test configuration suite covers $x\%$ more t-wise interactions than another one. Then, it is guaranteed that the other test configuration suite will miss all the faults contained in these x interactions. Finally, recent studies [6], [16] demonstrate the correlation between t-wise and fault detection.

9 Conclusion

T-wise testing aims at finding faulty feature interactions. However, full t-wise testing is hard and scalability is an issue: no approach is able to deal with high values of t (≥ 3) for large SPL FMs in a reasonable amount of time (in days). Moreover, there is no suitable technique supporting the generation of a fixed number of product configurations, according to a limited budget. This paper tackled these problems by proposing (a) approaches to prioritize product configurations while maximizing the t-wise coverage and (b) a scalable and flexible search-based technique to generate configurations under budget and time constraints for large FMs.

Our experiments, performed on 100 artificially generated and 14 real feature models from $t = 2$ to $t = 6$ show the feasibility and the scalability of our solutions. We managed to deal with the largest feature models available, such as the Linux kernel ($\approx$ 7,000 features, $\approx$ 200,000 constraints and $\approx$ 8.71E21 valid 6-sets) with up to 90.671% of 6-wise coverage achieved with 1,000 configurations. Thus, by enabling a partial but scalable t-wise coverage and by introducing flexibility in the testing process, our approaches pave the way to a potentially t-unrestricted combinatorial interaction testing.

REFERENCES

[1] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, 2001.

[2] D. Benavides, S. Segura, and A. R. Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, 2010.

[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Tech. Rep., November 1990.

[4] J. D. McGregor, "Testing a software product line," Tech. Rep., 2007.

[5] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Trans. Software Eng.*, vol. 23, no. 7, pp. 437–444, 1997.

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

[7] S. Oster, F. Markert, and P. Ritter, "Automated incremental pairwise testing of software product lines," in *SPLC*, 2010, pp. 196–210.

[8] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. L. Traon, "Pairwise testing for software product lines: comparison of two approaches," *Software Quality Journal*, vol. 20, no. 3-4, pp. 605–643, 2012.

[9] M. F. Johansen, Ø. Haugen, and F. Fleurey, "Properties of realistic feature models make combinatorial testing of product lines feasible," in *MoDELS*, 2011, pp. 638–652.

[10] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. L. Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *ICST*, 2010, pp. 459–468.

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

[12] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: a survey," *Softw. Test., Verif. Reliab.*, vol. 15, no. 3, pp. 167–199, 2005.

[13] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, p. 11, 2011.

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

[15] E. Reisner, C. Song, K.-K. Ma, J. S. Foster, and A. Porter, "Using symbolic evaluation to understand behavior in configurable software systems," in *ICSE (1)*, 2010, pp. 445–454.

[16] J. Petke, S. Yoo, M. B. Cohen, and M. Harman, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in *FSE*, 2013, pp. 26–36.

[17] M. Steffens, S. Oster, M. Lochau, and T. Fogdal, "Industrial evaluation of pairwise SPL testing with moso-polite," in *VaMoS*, 2012, pp. 55–62.

[18] M. F. Johansen, Ø. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *SPLC (1)*, 2012, pp. 46–55.

[19] C. Song, A. Porter, and J. S. Foster, "itree: Efficiently discovering high-coverage configurations using interaction trees," in *ICSE*, 2012, pp. 903–913.

[20] D. Marijan, A. Gotlieb, S. Sen, and A. Hervieu, "Practical pairwise testing for software product lines," in *SPLC*, 2013, pp. 227–235.

[21] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *VaMoS*, 2013, p. 7.

[22] K. Pohl, G. Böckle, and F. van der Linden, *Software product line engineering - foundations, principles, and techniques*. Springer, 2005.

[23] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," in *ESEC/SIGSOFT FSE*, 2013, pp. 290–300.

[24] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *ICSE*, 2011, pp. 461–470.

[25] M. F. Johansen, Ø. Haugen, F. Fleurey, E. Carlson, J. Endresen, and T. Wien, "A technique for agile and automatic interaction testing for product lines," in *ICTSS*, 2012, pp. 39–54.

[26] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *ISSTA*, 2007, pp. 129–139.

[27] A. Classen, P. Heymans, and P.-Y. Schobbens, "What's in a feature: A requirements engineering perspective," in *FASE*, 2008, pp. 16–30.

[28] A. M. Frisch, T. J. Peugniez, A. J. Doggett, and P. Nightingale, "Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings," in *J. Autom. Reasoning*, 2005, vol. 35, no. 1-3, pp. 143–179.

[29] Y. Lei and K.-C. Tai, "In-parameter-order: A test generation strategy for pairwise testing," in *HASE*, 1998, pp. 254–261.

[30] A. Calvagna and A. Gargantini, "A logic-based approach to combinatorial testing with constraints," in *TAP*, 2008, pp. 66–83.

[31] A. Hervieu, B. Baudry, and A. Gotlieb, "Pacogen: Automatic generation of pairwise test configurations from feature models," in *ISSRE*, 2011, pp. 120–129.

[32] M. B. Cohen, M. B. Dwyer, and J. Shi, "Coverage and adequacy in software product line testing," in *ROSATEA*, 2006, pp. 53–63.

[33] C. Yilmaz, M. B. Cohen, and A. A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *IEEE Trans. Software Eng.*, vol. 32, no. 1, pp. 20–34, 2006.

[34] R. C. Bryce and C. J. Colbourn, "One-test-at-a-time heuristic search for interaction test suites," in *GECCO*, 2007, pp. 1082–1089.

[35] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *ISSTA*, 2009, pp. 201–212.

[36] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru, "Prioritizing user-session-based test cases for web applications testing," in *ICST*, 2008, pp. 141–150.

[37] R. C. Bryce and A. M. Memon, "Test suite prioritization by interaction coverage," in *DOSTA*, 2007, pp. 1–7.

[38] R. C. Bryce, S. Sampath, J. B. Pedersen, and S. Manchester, "Test suite prioritization by cost-based combinatorial interaction coverage," *Int. J. Systems Assurance Engineering and Management*, vol. 2, no. 2, pp. 126–134, 2011.

[39] E. Uzuncaova, D. Garcia, S. Khurshid, and D. S. Batory, "Testing software product lines using incremental test generation," in *ISSRE*, 2008, pp. 249–258.

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

[41] F. Ensan, E. Bagheri, and D. Gasevic, "Evolutionary search-based test generation for software product line feature models," in *CAiSE*, 2012, pp. 613–628.

[42] D. Rubenstein, L. Osterweil, and S. Zilberstein, "An anytime approach to analyzing software systems," in *Proceedings of the 10th International Florida Artificial Intelligence Research Symposium*, 1997, pp. 386–391.

[43] J. D. McGregor and K. Im, "The implications of variation for testing in a software product line," pp. 59–64, 2007.

[44] P. Runeson and E. Engström, "Software product line testing - a 3d regression testing problem," in *ICST*, 2012, pp. 742–746.

[45] E. Engström and P. Runeson, "Software product line testing - a systematic mapping study," *Information & Software Technology*, vol. 53, no. 1, pp. 2–13, 2011.

[46] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *Information & Software Technology*, vol. 53, no. 5, pp. 407–423, 2011.

[47] D. Le Berre and A. Parrain, "The sat4j library, release 2.2, system description," *Journal on Satisfiability, Boolean Modeling and Computation(JSAT)*, vol. 7, pp. 59–64, 2010.

[48] H. Hemmati and L. C. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *ISSRE*, 2010, pp. 141–150.

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

[50] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+ 1) evolutionary algorithm," *Theor. Comput. Sci.*, vol. 276, no. 1-2, pp. 51–81, Apr. 2002.

[51] M. Harman, Y. Jia, and W. B. Langdon, "Strong higher order mutation-based test data generation," in *FSE*, 2011, pp. 212–222.

[52] M. Mendonça, A. Wasowski, and K. Czarnecki, "Sat-based analysis of feature models is easy," in *SPLC*, 2009, pp. 231–240.

[53] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Trans. Software Eng.*, vol. 36, no. 6, pp. 742–762, 2010.

[54] M. Mendonça, M. Branco, and D. D. Cowan, "S.P.L.O.T.: software product lines online tools," in *OOPSLA Companion*, 2009, pp. 761–762.

[55] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Softw. Test., Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, 2012.

[56] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of acts: A case study," in *ICST*, 2012, pp. 591–600.

[57] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, "Augmenting simulated annealing to build interaction test suites," in *ISSRE*, 2003, pp. 394–405.

[58] A. Arcuri and L. C. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *ICSE*, 2011, pp. 1–10.

[59] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Software Eng.*, vol. 32, no. 9, pp. 733–752, 2006.

[60] C. H. P. Kim, D. S. Batory, and S. Khurshid, "Reducing combinatorics in testing product lines," in *AOSD*, 2011, pp. 57–68.

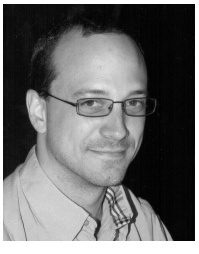
[61] M. F. Johansen, Ø. Haugen, F. Fleurey, A. G. Eldegard, and T. Syversen, "Generating better partial covering arrays by modeling weights on sub-product lines," in *MoDELS*, 2012, pp. 269–284.

[62] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Multi-objective test generation for software product lines," in *SPLC*, 2013, pp. 62–71.

[63] B. C. S. S. I. G. on Software Testing, *Standard for Software Component Testing*.

[64] P. G. Frankl, R. G. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Trans. Software Eng.*, vol. 24, no. 8, pp. 586–601, 1998.

Christopher Henard is a Ph.D. student at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg. He received his M.Sc. in Computer Science in 2011 from Télécom Nancy, (former Ecole Supérieure d'Informatique et Applications de Lorraine), an engineering school at the University of Lorraine, France. His research focuses on software product lines.



Mike Papadakis received his B.Sc, M.Sc. and Ph.D degrees in Computer Science from the Athens University of Economics and Business (Greece). He is research associate at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg. His research interests include software product lines, software testing and software debugging.

Gilles Perrouin is currently an FNRS post-doctoral researcher at the Faculty of Computer Science, University of Namur, Belgium. He is a member of the PReCISE research centre, in which he explores software product lines, modeling and model-based testing. Gilles Perrouin also serves regularly as a (co)-referee for journals and conferences and is an IEEE member. Gilles Perrouin holds a joint Ph.D. degree from the University of Luxembourg and Namur.

Jacques Klein is research scientist at the University of Luxembourg and at the Interdisciplinary Centre for Security, Reliability and Trust (SnT). He received a Ph.D. degree in Computer Science from the University of Rennes, France in 2006. His main areas of expertise are software security by applying software engineering to security; model-driven engineering, with a focus on model composition and model@runtime; software testing, and software product lines.

Patrick Heymans is professor of software engineering at the University of Namur, Belgium. He is founding member and current director of the PReCISE research centre where he leads the requirements engineering and software product line efforts. His main research interests are in the application of formal and visual modeling techniques to improve the quality of software products and processes.

Yves Le Traon is professor at the University of Luxembourg, in the domain of software engineering, reliability, testing and security. His current research interests include and combine software product line re-engineering and testing, Android security, mutation testing, model-driven security, and search-based software engineering. He received a Ph.D. degree in Computer Science at the Institut National Polytechnique in Grenoble, France, in 1997. He was associate professor in France at the University of Rennes, and then full professor at Télécom Bretagne until he reaches Luxembourg in 2009. He is currently the head of the CSC Research Unit (Department of Computer Science) and an active member of the Interdisciplinary Centre for Security, Reliability and Trust (SnT), where he leads the SERVAL group (SEcuRity design and VALidation). He has been on the program, steering, or organization committees of many international IEEE software engineering conferences. He also belongs to the steering committee of IEEE ICST. He is a member of the IEEE Computer Society.