Comparative Evaluation of Similarity-Based Prioritization Techniques in Search-Based Test Case Generation for Software Product Lines

Pierre-Louis Clavel[1] and Camille Salinesi[1]

[1]University of Paris 1 - Sorbonne, 90, Rue de Tolbiac, 75013 Paris, France

Mars 2025

Abstract

Testing software product lines (SPLs) is a challenging issue due to the combinatorial explosion of valid product configurations. To address this, similarity-based prioritization techniques are used to select diverse and representative subsets of configurations, maximizing feature interaction coverage. Although search-based test case generation has been widely studied, few empirical studies have compared prioritization techniques in such contexts. This paper presents a controlled experiment that compares five prioritization techniques: Novelty Score, Enhanced Jaro-Winkler (combined with Local and Global Maximum Distance algorithms), Dice-Jaro-Winkler, and a state-of-the-art baseline technique that is based on Jaccard distance. All techniques were implemented in the PLEDGE tool and applied to the same initial test case samples to neutralize generation variability. The evaluation was conducted across nine feature models of varying sizes and three predefined test suite sizes. Results show that the Novelty Score technique consistently achieves higher pairwise coverage than all other methods. Enhanced Jaro-Winkler combined with Global Maximum Distance yields competitive results but exhibits performance variability depending on the feature model and test suite size. In contrast, the Dice-Jaro-Winkler technique underperforms across most scenarios. This study provides empirical evidence supporting the integration of Novelty Score-based prioritization in SPL testing and proposes an experimental framework reusable for future comparative studies.

Keywords: Software product line testing, test case generation, similarity-based prioritization, search-based testing

1 Introduction

Software Product Line Engineering (SPLE) has emerged as a major paradigm in software development, addressing the need for mass customization through systematic reuse [16]. A software product line (SPL) is defined as a set of software-intensive systems that share a common set of features and are developed from a shared set of core assets in a prescribed way [6]. While SPLE improves efficiency and cost-effectiveness, it also introduces new challenges in quality assurance, particularly in testing. Due to feature variability and constraints, the number of possible product configurations increases exponentially with the number of features [9], making exhaustive testing intractable.

Feature models are commonly used to represent SPLs by capturing both commonalities and variabilities among products [4]. A product corresponds to a valid feature configuration that satisfies all constraints in the model. As the number of valid configurations grows rapidly, selecting a representative subset of products to test becomes critical. Test case selection strategies, such as t-wise testing [20] [21] [13], aim to maximize interaction coverage among features. However, the combinatorial explosion of t-wise interactions results in high computational demands, making traditional exhaustive sampling approaches impractical.

To address this, search-based test case generation approaches have been introduced, leveraging metaheuristic algorithms to generate diverse configurations under time and resource constraints. In particular, similarity-based prioritization techniques [23] have gained attention as a means to enhance diversity in the final test suite by selecting dissimilar products. These techniques order test cases based on similarity distances or diversity metrics, often integrated within a search-based generation algorithm. In the approach proposed by Henard *et al.*, products are prioritized using dissimilarity measures such as the Jaccard distance, and selected via a fitness function that guide the search toward diverse test suites [9].

Building on this foundation, several alternative prioritization techniques have emerged. The Novelty Score, proposed by Xiang *et al.*, evaluates a product's distinctiveness based on its distance from its nearest neighbors [26]. Other studies have explored enhanced distance metrics, such as the Enhanced Jaro-Winkler distance, which incorporates a Degree of Difference to account for deselected features [1], and the Dice-Jaro-Winkler hybrid metric, which combines multiple similarity dimensions [24]. While promising, these techniques have not yet been extensively compared in a controlled and homogeneous experimental setting within a search-based SPL testing framework.

This paper addresses this gap through a comparative empirical study that evaluates the impact of several similarity-based prioritization techniques on pairwise coverage, a widely used metric for assessing test suite diversity in SPLs. The following research questions guide our investigation:

RQ: Which similarity-based prioritization technique achieves the highest pairwise coverage when applied in a search-based test case generation framework for SPLs?

SRQ1: How effective is the Novelty Score prioritization technique in achieving pairwise coverage?

SRQ2: How effective is the Enhanced Jaro-Winkler distance metric when integrated with Local and Global Maximum Distance prioritization strategies?

SRQ3: What is the performance of the Dice-Jaro-Winkler prioritization algorithm for pairwise coverage compared to the state-of-the-art approach?

To answer these questions, we conduct an experimental comparison of five prioritization techniques: Novelty Score, Enhanced Jaro-Winkler combined with Local and Global Maximum Distance algorithms, Dice-Jaro-Winkler, and the baseline technique proposed by Henard *et al.*, which uses the Local Maximum

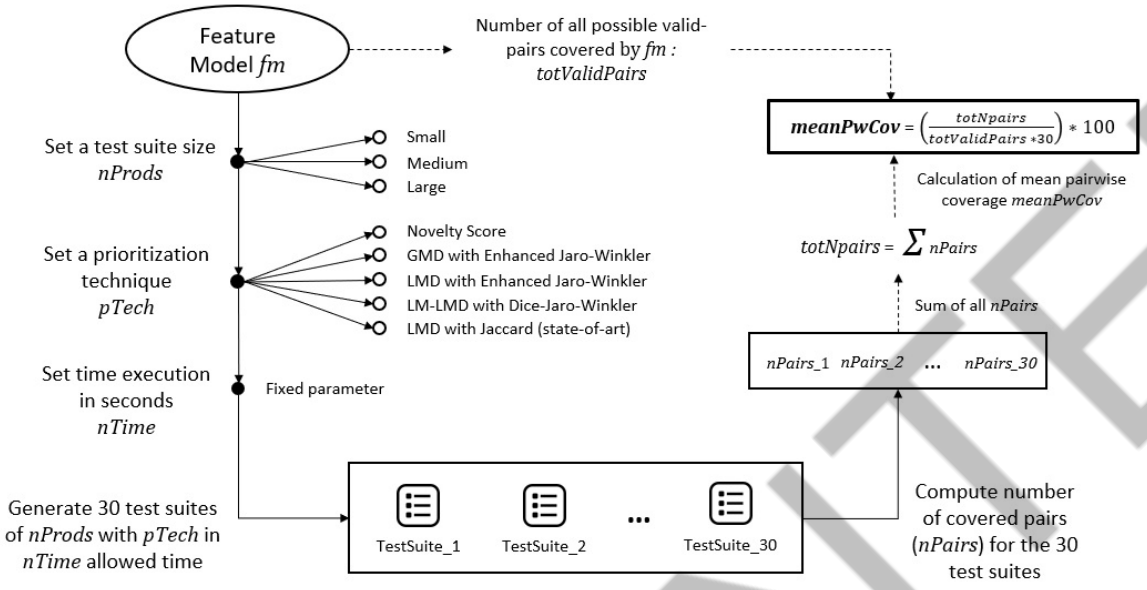

Figure 1: Experimental framework of mean pairwise coverage extraction

Distance algorithm with Jaccard distance [9]. All techniques are implemented within the PLEDGE tool [11] and applied to the same initial test suites across nine feature models and three test suite sizes.

Our contributions are threefold: (1) an empirical evaluation of contemporary prioritization techniques in SPL testing, (2) a reusable experimental framework for comparative studies, and (3) empirical insights into the performance of these techniques across varying model sizes and test suite dimensions. This work contributes to both the theoretical understanding and practical application of similarity-based test case prioritization in software product line engineering.

2 Method

2.1 Experimental design

2.1.1 Experimental framework

To compare the selected prioritization techniques, an experimental framework (Figure 1), inspired by Xiang *et al.*'s experimental design, was developed to extract the mean pairwise coverage of the studied techniques applied to various feature models and test suite sizes. To produce quantitative results, all test case generations were executed 30 times, following Xiang *et al.*'s methodology. The same search-based test case generation process was used in this study, as detailed in Section 2.1.2. We ensured that, for a given feature model and test suite size, all executions started from a predefined set of products specific to each execution. This approach allowed us to use the same sets for all prioritization techniques, thereby minimizing the influence of unpredictable generation on the final comparison. To achieve this, thirty samples of test suites were generated for each feature model and test suite size using the SAT4j solver [3] and unpredictable generation [9].

An analysis is conducted on the performance of the techniques across various test suite sizes. Three categories of test suite sizes have been defined in this study, based on the feature model and the ability of unpredictable generation to achieve a certain range of pairwise coverage. This is detailed in Section

2.1.5. A fixed execution time, ranging from 6 to 30 seconds, was set for all executions on the same feature model.

2.1.2 Test case generation

A tool called PLEDGE, utilizing the search-based approach defined by Henard *et al.*, has been deployed [11]. This tool allows users to load a feature model [19] and execute the Evolutionary Algorithm by specifying the desired execution time and final test suite size. Besides the user-friendly interface of the tool, the MVC architecture programmed in Java helps developers to integrate their own test case generation or prioritization algorithms, allowing them to test their efficiency in terms of pairwise coverage. For this study, PLEDGE was used to execute test case generation with the Evolutionary Algorithm. However, instead of generating an unpredictable set of test cases for each execution, a predefined test suite was selected. The implementation work for this experiment involved designing prioritization techniques compatible with the search-based approach and integrating them into PLEDGE[1]. All these techniques are described in the following section.

2.1.3 Prioritization techniques

In Henard *et al.*'s search-based approach, prioritization techniques are defined by an algorithm that sorts products based on dissimilarity and a fitness function that evaluates the overall dissimilarity of the test suite. During the specified execution time, the Evolutionary Algorithm iterates over the test suite by mutating the last product in the list and generating a new unpredictable product. If this new product improves the fitness function, the prioritized test suite with the added product is retained; otherwise, the test suite from before the iteration is preserved for the remainder of the algorithm.

The PLEDGE tool already includes two algorithms defined as prioritization techniques: Local Maximum Distance (LMD) and Global Maximum Distance (GMD). The LMD approach selects the two products with the highest distance and places them first in the ordered test suite, then repeats the process without considering the already ordered products. In contrast, the GMD approach considers the ordered products and maximizes the distances between the remaining products and those already ordered. The default distance used for calculating dissimilarity between two products in prioritization algorithms is the Jaccard distance. For both approaches, the final fitness function is the sum of all calculated distances.

Novelty Score prioritization technique

Instead of calculating dissimilarity between two products, Xiang *et al.* proposed an approach that evaluates the dissimilarity of a single product within a set of products by calculating the novelty score. This metric is defined as the sum of the k-nearest distances divided by k, where k determines the number of product neighbors selected for evaluating the novelty score of a product.

In this study, we propose a new approach to the Novelty Score algorithm by redefining it as a prioritization algorithm (Algorithm 1) that takes an unprioritized list and the parameter k as inputs. After calculating all distances between products (lines 1 to 6), the novelty score is computed for all products (lines 7 to 10). The returned prioritized list of products is the initial list sorted by novelty scores.

In this experiment, we set k to the same default value as Xiang *et al.*, equal to the test suite size. The distance used is the Anti-Dice distance. Finally, the fitness function has been set as the sum of all novelty scores.

Enhanced Jaro-Winkler prioritization techniques

As done by Halim *et al.* in their experiment, the Enhanced Jaro-Winkler distance has been applied with Local Maximum Distance and Global Maximum Distance. Since these prioritization techniques are

[1] https://github.com/pierreLouisClv/PLEDGE_src/tree/main

Algorithm 1 Novelty Score Prioritization Algorithm

Require: $products = \{p_1, p_2, \ldots, p_n\}$, k
Ensure: $prioritizedProducts$
for $i = 1$ to $size$ **do**
  for $j = i + 1$ to $size$ **do**
    $distancesMatrix[i][j] \leftarrow \text{calculateAntiDiceDistance}(p_i, p_j)$
    $distancesMatrix[j][i] \leftarrow distancesMatrix[i][j]$
  end for
end for
for $i = 1$ to $size$ **do**
  $sortedDistances \leftarrow \text{sort}(distancesMatrix[i])$
  $noveltyScores[i] \leftarrow \frac{\sum_{j=1}^{k} sortedDistances[j]}{k}$
end for
$prioritizedProducts \leftarrow \text{sort}(products, \text{by descending } noveltyScores)$

already defined in PLEDGE, the only task was to implement the Enhanced Jaro-Winkler distance among the distances provided by the tool. This distance modifies the original Jaro-Winkler distance by defining a Degree of Difference that considers deselected features.

Dice Jaro-Winkler prioritization technique

The prioritization technique designed by Suleiman *et al.* has been implemented in this experiment, following the same approach as the authors' original study. In this algorithm, the authors reuse the Enhanced Jaro-Winkler distance and combine it with the Dice distance. Consequently, the prioritization technique is an enhancement of Local Maximum Distance algorithm. The key difference is that, after ordering the two products with the highest distance, the algorithm selects the next product based on the highest distance from the last ordered product. In this study, the fitness function is the same as that used in the Local Maximum Distance algorithm, returning the sum of all calculated distances.

2.1.4 Feature models selection

In this study, several feature models were selected to evaluate prioritization techniques across different feature model sizes, as presented in Table 1. Initially, these models were chosen from Xiang *et al.*'s study because the number of valid 2-sets has been published in their experiment. This value allows us to avoid the time-consuming computation of all valid 2-sets for a feature model and will be used for the final pairwise coverage calculation. Additionally, these models have accessible DIMACS files that can be processed in PLEDGE.

Three models with similar feature sizes, ranging from 40 to 50 features, were selected, including one with a large number of constraints (DSSample with 201 constraints). Furthermore, four moderate feature models, ranging from 79 to 290 features, were included in the experiment. All of these are real-world case studies. In contrast, two large feature models [2] [3] with 1,000 features, generated from SPLOT [18], were also selected. For all feature models, a fixed execution time ranging from 6 to 30 seconds was set. All feature models characteristics are presented in Table 1.

2.1.5 Test suite sizes definition

In addition to feature models, various test suite sizes were defined to demonstrate the efficiency of prioritization techniques across different test suite dimensions. A preliminary study was conducted to determine

[2]SPLOT-1 full name: SPLOT-3CNF-FM-1000-200-0,50-SAT-1
[3]SPLOT-2 full name: SPLOT-3CNF-FM-1000-200-0,50-SAT-7

Feature model	Features	Cons.	Free Features	Valid 2-Sets	Exec. Time (s)
DSSample	41	201	34	2592	6
WebPortal	43	68	39	3196	6
Drupal	48	79	40	3751	6
Amazon	79	250	73	10555	8
CocheEcologico	94	191	57	11075	10
Printers	172	310	122	42638	15
E-Shop	290	426	260	149723	20
SPLOT-1	1000	1875	949	1861476	30
SPLOT-2	1000	1874	967	1919404	30

Table 1: Feature models characteristics

the values of the test suite size parameter used in this experiment. This approach enhances the final comparison by ensuring that all executions, regardless of the loaded feature model, begin with a similar pairwise coverage for small, medium, and large test suites resulting from the initially generated sample.

A test suite size can be approximated by setting a pairwise coverage threshold and executing the following steps:

1. Load a feature model.
2. Generate 30 random test suites for test suite sizes ranging from 3 to 100.
3. Compute the mean pairwise coverage for each test suite size.
4. Select the smallest test suite size where the mean pairwise coverage reaches the defined threshold.

For all feature models, these steps were followed for three thresholds: 78%, 84%, and 92%, to define three categories of test suite sizes: small, medium, and large.

Step 4 is based on the hypothesis that pairwise coverage increases with the test suite size, under the theoretical assumption that adding another product to a test suite may enhance feature interaction coverage.

However, the final obtained values for test suite sizes remain approximations due to the variability inherent in sample generation.

2.2 Pairwise coverage extraction

T-wise coverage is calculated by determining the number of interactions between t-features covered in the entire test suite, divided by the total number of valid t-sets in the feature model. As explained in Section 2.1.4, it was not necessary to compute all valid 2-sets. For calculating pairwise coverage, the only required computation was enumerating all pairs covered by the final test suite. To achieve this, a Python script utilizing the itertools library was employed to find combinations between two features, which proved to be more efficient than the solution proposed in PLEDGE for pairwise computation.

3 Experimental results

3.1 Presentation of results

3.1.1 Approximation of test suite sizes

The results of the preliminary study on test suite size approximation are presented in Table 2. These results will be utilized in the main study of this paper as generation parameters for small, medium, and large test suite sizes.

Feature Models	**Test suite sizes**		
	Small	**Medium**	**Large**
DSSample	16	23	46
WebPortal	6	8	16
Drupal	6	7	11
Amazon	31	47	78
CocheEcologico	6	9	17
Printers	6	9	21
E-Shop	6	8	13
SPLOT-1	13	20	43
SPLOT-2	9	12	22

Table 2: Small, medium and large test suite sizes defined for feature models

3.1.2 Extraction of pairwise coverages

The pairwise coverage extraction was performed on all feature models, as shown in Figure 1, by generating 30 test suites with all prioritization techniques across all test suite sizes. The results are presented in Table 3. Executions and pairwise computations were performed on a system equipped with a 13th Gen Intel(R) Core(TM) i7-13620H CPU @ 2.40 GHz, 16 GB RAM, and running Windows 11.

The involved techniques are indicated as follows, with their respective column names in Table 3:

1. Novelty Score : *NS*
2. Global Maximum Distance with Enhanced Jaro-Winkler distance : *EJW (GMD)*
3. Local Maximum Distance with Enhanced Jaro-Winkler distance : *EJW (LMD)*
4. LM-LMD with Dice-Jaro-Winkler distance : *DJW*
5. State-of-the-art - Traditional Local Maximum Distance with Jaccard distance : *JAC (LMD)*

All pairwise coverages extracted from the generated samples used for the initialization of the test suite are indicated in the column *Initial Set*.

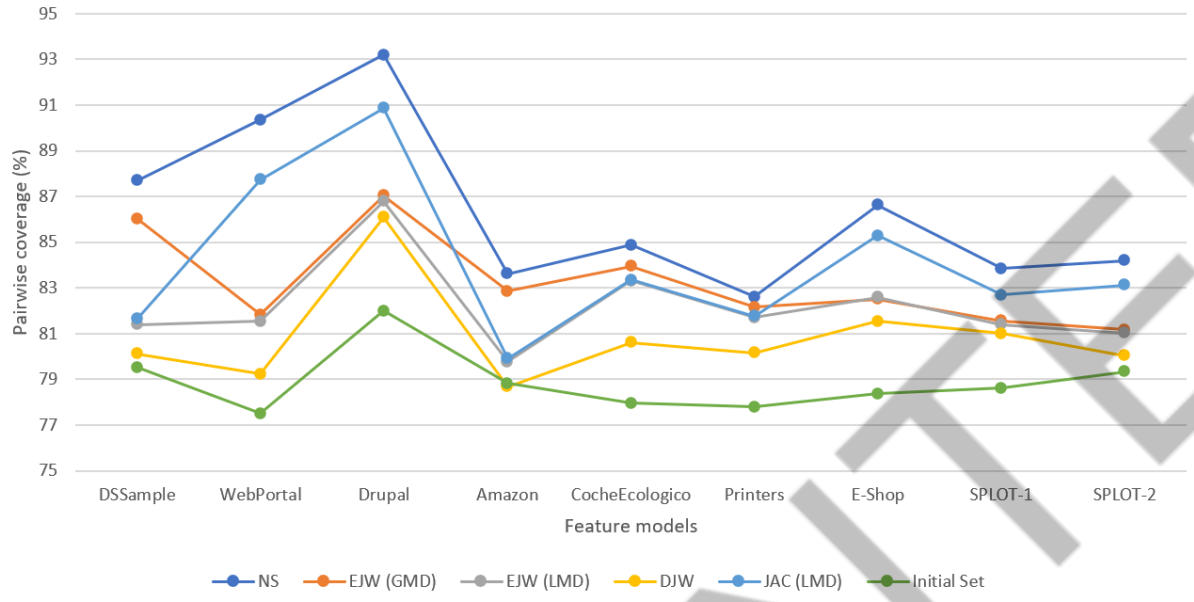

Figure 2: Mean pairwise coverages of small test suite sizes

3.2 Analysis

Figures 2, 3, and 4 illustrate the results from Table 3 for small, medium, and large test suite sizes, respectively. Each graph provides an overview of the performance of the prioritization techniques compared to each other, including the state-of-the-art approach, Jaccard distance applied with Local Maximum Distance (in light blue). The indicated pairwise coverage of the unpredictable generated set (in green) serves as a reference for evaluating the improvement in pairwise coverage achieved by the prioritization techniques. As explained in Section 2.3.4, all prioritization techniques start with the same predefined sets.

Based on the experimental results, we address the studied research questions.

SRQ1: How effective is the Novelty Score prioritization technique in achieving pairwise coverage?

The Novelty Score prioritization technique (NS) demonstrates a significant average pairwise coverage improvement of 7.48 percentage points for small test suite sizes, 6.05 percentage points for medium sizes, and 2.98 percentage points for large sizes, compared to the initial set of products.

Compared to the state-of-the-art approach, NS exhibits superior pairwise coverage performance across all feature model case studies and test suite sizes. The prioritization technique achieves an average improvement of more than 2 percentage points for small (2.29 percentage points) and medium (2.30 percentage points) test suite sizes, and 1.35 percentage points for large sizes.

SRQ2: How effective is the Enhanced Jaro-Winkler distance metric when integrated with Local and Global Maximum Distance prioritization strategies?

The Enhanced Jaro-Winkler distance, when applied with Global Maximum Distance (GMD) and Local Maximum Distance (LMD), improves the initial pairwise coverage in all cases, except for the WebPortal and SPLOT-2 feature models with large test suite sizes. For the SPLOT-2 case study, the decrease in pairwise coverage for both prioritization techniques is minimal (0.38 percentage points for GMD and 0.45

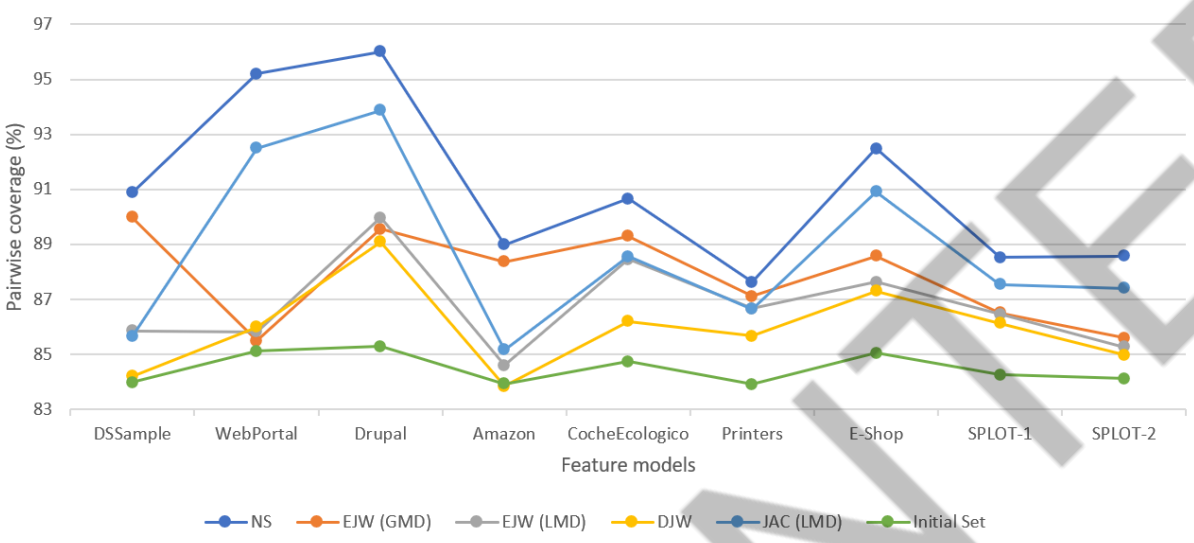

Figure 3: Mean pairwise coverages of medium test suite sizes

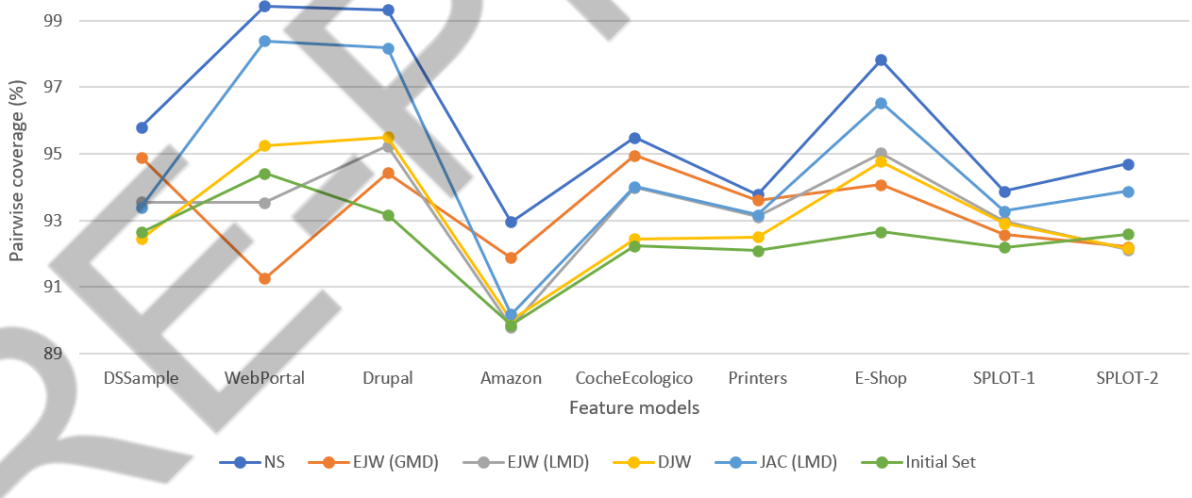

Figure 4: Mean pairwise coverages of large test suite sizes

for LMD). However, for the WebPortal case study, the decrease is more significant for GMD, which loses 3.16 percentage points compared to 0.88 for LMD.

Despite this, the GMD technique achieves a better average improvement in percentage points across all case studies for large test suite sizes, with an improvement of 0.89 percentage points compared to 0.83 for LMD. This trend is also observed for other test suite sizes, with GMD showing an average improvement of 3.34 percentage points compared to 2.36 for LMD in medium sizes, and 4.35 percentage points compared to 3.29 for LMD in small sizes.

Referring to the state-of-the-art approach, the GMD technique shows notably better pairwise coverage performance in four of the nine case studies (DSSample, Amazon, CocheEcologico, and Printers).

Conversely, the LMD technique shows similar results for CocheEcologico and Printers, with the state-of-the-art approach showing a slight improvement that never exceeds 0.1 percentage points for any test suite size. For the DSSample feature model, the results are also close. LMD shows a small improvement of about 0.21 percentage points for medium test suite sizes and about 0.15 percentage points for large sizes, but it decreases by about 0.24 percentage points for small sizes. The state-of-the-art approach shows better performance in all other case studies across all test suite sizes.

When combining all feature models, the state-of-the-art approach shows a better average improvement than GMD for all test suite sizes, with improvements of 0.82 percentage points for small sizes, 0.86 for medium sizes, and 1.24 for large sizes. These results are heavily influenced by the differences observed in the WebPortal case study, where GMD performed 5.92 percentage points worse than the state-of-the-art for small test suites, 7.01 percentage points worse for medium sizes, and 7.12 for large sizes.

SRQ3: What is the performance of the Dice-Jaro-Winkler prioritization algorithm for pairwise coverage compared to the state-of-the-art approach?

The Dice-Jaro-Winkler prioritization technique (DJW) demonstrates a small average pairwise coverage improvement of 1.95 percentage points for small test suite sizes, 1.45 percentage points for medium sizes, and 0.68 percentage points for large sizes, compared to the initial set of products. It decreases the initial pairwise coverage in four cases (Amazon feature model for small and medium test suite sizes, DSSample for large sizes, and SPLOT-2 for large sizes).

Compared to the state-of-the-art approach, DJW exhibits inferior pairwise coverage performance across all feature model case studies and test suite sizes. The state-of-the-art approach achieves an average improvement of 3.21 percentage points for small test suite sizes, 2.76 percentage points for medium sizes, and 1.45 percentage points for large sizes, compared to DJW. No case in the experiment shows better pairwise coverage from DJW, indicating the limitations of the algorithm for pairwise coverage improvement.

RQ: Which similarity-based prioritization technique achieves the highest pairwise coverage when applied in a search-based test case generation framework for SPLs?

Even for the feature models where EJW (GMD) showed better results than the state-of-the-art approach, mean pairwise coverages are still lower than those achieved by NS. The Novelty Score prioritization technique outperforms other techniques across all case studies and test suite sizes. In terms of mean pairwise coverage, the state-of-the-art approach performs better on average than EJW (GMD), and this average difference grows as the test suite size increases.

Although EJW (LMD) improves initial mean pairwise coverage in more than 90% of the comparison cases, it never shows notable improvement against the state-of-the-art approach. This observation is even more pronounced for the Dice-Jaro-Winkler algorithm.

3.3 Discussion

In this section, we further investigate the previously observed results by analyzing the characteristics of feature models and the relevance of similarity-based fitness functions.

Feature model complexity. The approximation of test suite sizes (Table 2) reveals significant variability in the number of randomly generated products required to achieve high pairwise coverage across different feature models. For instance, in the Drupal case study, only 11 randomly generated products are needed to reach an average pairwise coverage of 92%, whereas 78 products are required for the Amazon case study. This disparity reflects a substantial difference in search complexity, indicating that dissimilar products and uncovered pairs are more challenging to identify in more complex feature models such as Amazon.

Tables 1 and 2 clearly demonstrate that there is no direct correlation between the number of features and feature model complexity. Similarly, a comparison of test suite sizes between the SPLOT-1 and SPLOT-2 case studies—both of which contain the same number of features and differ by only one constraint—suggests that there is no clear correlation between the number of constraints and model complexity either. These observations indicate that complexity is influenced by the composition of constraints, which is not explicitly detailed in Table 1, as this study does not focus on constraint composition.

Relevance of similarity-based fitness functions. Xiang et al. conducted a correlation analysis showing that the novelty score metric has a significant positive correlation with t-wise coverage [26]. Our experimental results confirm the effectiveness of the similarity-based fitness function proposed by Xiang et al. in the context of Evolutionary Algorithm-based search, particularly for improving pairwise coverage.

For the other prioritization techniques, we define the fitness functions as the sum of distance metrics between all products, following the approach of Henard et al. for the Jaccard distance. While this strategy has been shown to be effective for t-wise coverage using the Jaccard distance, its suitability for the Enhanced Jaro-Winkler (EJW) and Dice-Jaro-Winkler (DJW) distances remains uncertain based on our experimental findings. An unsuitable fitness function fails to adequately represent the dissimilarity of the test suite, potentially eliminating highly dissimilar products while retaining overly similar ones, ultimately affecting pairwise coverage improvement.

Focusing on the WebPortal case study, the application of the EJW distance with both Global Maximum Distance (GMD) and Local Maximum Distance (LMD) prioritization techniques leads to a decline in pairwise coverage for medium and large test suites. This result highlights the inadequacy of the fitness function, which is identical for both techniques. This limitation is particularly pronounced for the GMD strategy, which exhibits an even greater decrease in pairwise coverage as the test suite size increases.

Overall, the EJW (GMD) approach demonstrates lower efficiency compared to the state-of-the-art method for less complex feature models (WebPortal, Drupal, E-Shop, and SPLOT-2). However, it yields promising results for two of the three most complex feature models (DSSample and Amazon). A deeper investigation into the composition of feature model constraints could provide further insights into the relationship between EJW (GMD) performance and model complexity.

Further research is needed to identify the most appropriate fitness function for the EJW distance, in order to fully leverage its potential in search-based test case generation.

4 Threats to Validity

In this section, threats to validity are discussed.

Internal Validity. Possible bias could exist in the implementation of prioritization techniques. To mitigate this, we ensured that the code adheres exactly to the algorithms by using debugging tools and testing the prioritization techniques on small feature models. The code for the prioritization techniques,

pairwise computation, result collection, and all feature models is available in the following GitHub repository: https://github.com/pierreLouisClv/PLEDGE_src/tree/main. We also ensured that prioritization techniques were compared equivalently by using the same tool, PLEDGE, and the same generation algorithm.

The Evolutionary Algorithm uses unpredictably generated products as the initial set for test case generation. This random approach can produce variable sets of products in terms of pairwise coverage, potentially influencing the final results. To mitigate this, we predefined sets of products to ensure that test case generation initializes execution with the same sets for all prioritization techniques. Additionally, we reduced the impact of the unpredictable approach by generating 30 executions and extracting the mean pairwise coverage for all experimental cases.

External Validity. This study produces quantitative results for the comparison across diverse feature models ranging from 41 to 1,000 features and across three categories of test suite sizes. This approach mitigates the threat posed by feature model singularities and variations in user-defined test suite size parameters. However, the results cannot be generalized to feature models exceeding 1,000 features, as this experiment did not include very large feature models due to the consideration of limiting the total execution time.

5 Related Works

Test case generation can be managed by several sampling methods. In recent years, numerous combinatorial sampling algorithms have been developed based on t-wise interactions. The Chvatal algorithm [5] [13] optimizes feature interactions by approximating minimal covering arrays. An improved algorithm, ICPL [12], has been proposed but does not scale for higher values of t = 3. The minimization of covering arrays has also been addressed by algorithms such as MoSo-PoLiTe [20], IncLing [8], CASA [7], and YASA [14]. Recently, a local search method, LS-Sampling-Plus [15], has been proposed to maximize t-wise coverage for a defined test suite size. Additionally, cost-effective search-based test case selection methods have been developed, including multi-objective algorithms that optimize both pairwise coverage and cost [25] [10].

These methods are often time-consuming. This study explores techniques that can produce high t-wise coverage within a predefined allowed time and test suite size. Theses resource constraints have been addressed by Henard *et al.* as user-defined parameters using a search-based approach (Evolutionary Algorithm) that starts with an initial set of products generated unpredictably [9]. This type of random sampling avoids time-consuming t-wise computations and makes the technique scalable to larger feature models compared to traditional sampling algorithms [17]. The efficiency of final t-wise coverage depends on user-defined resource parameters and the prioritization technique used for prioritizing products and calculating a fitness function. This search-based approach has been modified by Xiang *et al.*, who use an unpredictable initial set but define a probabilistic test case selection for the search process [26]. This study employs the Evolutionary Algorithm's approach for the search-based method and evaluates the technique in terms of pairwise coverage for different similarity-based prioritization techniques.

Similarity-based prioritization techniques have emerged in recent years to select diverse products. Diversity is often evaluated using the Early Fault Detection approach. Some studies [2] [22] used this approach to compare similarity distances with different prioritization techniques by prioritizing a test case sample and calculating the APFD metric. Similarly, the Enhanced Jaro-Winkler distance [1] and Dice-Jaro-Winkler algorithms [24] have been evaluated using APFD. As t-wise coverage is widely used for evaluating test suite diversity, it can serve as a metric for evaluating the Enhanced Jaro-Winkler distance and Dice-Jaro-Winkler algorithms within the Evolutionary Algorithm search-based approach. This study conducts this comparison and proposes a prioritization approach using the Novelty Score algorithm by reusing the novelty score metric in a prioritization technique, which was not addressed by Xiang *et al.*

Overall, there is a lack of test case generation experiments that compare prioritization techniques in a

search-based context. This paper provides an applicable method for comparing prioritization techniques with pairwise coverage metric.

6 Conclusion and future works

This paper presented a comparative empirical study of similarity-based prioritization techniques applied in a search-based test case generation framework for Software Product Lines (SPLs). The objective was to assess the impact of these techniques on pairwise coverage, a key metric for evaluating the diversity and effectiveness of test suites in SPL testing. The study addressed a gap in the literature by providing a controlled and reproducible evaluation framework, and by systematically comparing five prioritization strategies across nine feature models and three categories of test suite sizes.

In response to our main research question (RQ), the study shows that prioritization techniques have a significant influence on the effectiveness of search-based test case generation in SPL contexts. Among the techniques evaluated, the Novelty Score clearly stands out as the most effective method for maximizing pairwise coverage, across all feature models and test suite sizes.

Regarding SRQ1, our results confirm the strong performance of the Novelty Score prioritization technique. It consistently achieves higher pairwise coverage than both the state-of-the-art baseline and other contemporary techniques. These results support the integration of Novelty Score as a core component of test suite prioritization strategies in SPL testing.

As for SRQ2, the Enhanced Jaro-Winkler distance, when used in conjunction with the Global Maximum Distance algorithm, demonstrates promising results in several case studies. However, its performance remains variable and sensitive to the structural characteristics of the feature models. When combined with the Local Maximum Distance algorithm, the technique only yields modest improvements and generally fails to outperform the baseline approach.

SRQ3 is not answered positively: the Dice-Jaro-Winkler prioritization technique does not show significant benefits in terms of pairwise coverage. It even sometimes underperforms compared to the state-of-the-art technique and, in some cases, leads to degraded coverage. This suggests that combining multiple similarity metrics does not automatically improve prioritization effectiveness and may introduce undesirable side effects.

Beyond these empirical findings, the study has practical implications for software testing engineers and SPL practitioners. First, the results highlight the importance of carefully selecting prioritization strategies when using search-based generation techniques, as suboptimal choices can substantially reduce test suite diversity. Second, the integration of the Novelty Score into existing testing frameworks such as PLEDGE offers a concrete and effective solution for improving coverage without increasing computational cost or test suite size. Third, the experimental methodology developed in this study can be reused by practitioners to benchmark additional prioritization techniques or adapt selection strategies to specific project constraints.

Future work could explore alternative diversity metrics beyond pairwise coverage, investigate the effect of prioritization on fault detection capability, or develop adaptive prioritization strategies sensitive to model-specific features. The evaluation of multi-objective approaches that consider both test coverage and testing cost also constitutes a promising direction. Finally, expanding the analysis to very large-scale feature models could provide further insights into the scalability of the tested techniques.

By combining rigorous empirical analysis with actionable recommendations, this study contributes to the advancement of search-based software testing and provides practical guidance for improving test suite effectiveness in industrial software product lines.

References

[1] Shahliza Abd Halim, Dayang Norhayati Abang Jawawi, and Muhammad Sahak. "Similarity distance measure and prioritization algorithm for test case prioritization in software product line testing". In: *Journal of Information and Communication Technology* 18.1 (2019), pp. 57–75.

[2] Safwan Abd Razak, Mohd Adham Isa, and Dayang Norhayati Abang Jawawi. "A Comparison on Similarity Distances and Prioritization Techniques for Early Fault Detection Rate". In: *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 9.3-3 (2017), pp. 89–94.

[3] Parrain Anne et al. "The SAT4J library release 2.2 system description". In: *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)* 7 (2010), pp. 59–64.

[4] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. "Automated analysis of feature models 20 years later: A literature review". In: *Information systems* 35.6 (2010), pp. 615–636.

[5] Vasek Chvatal. "A greedy heuristic for the set-covering problem". In: *Mathematics of operations research* 4.3 (1979), pp. 233–235.

[6] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Boston: Addison-Wesley, 2001.

[7] Brady J Garvin, Myra B Cohen, and Matthew B Dwyer. "Evaluating improvements to a meta-heuristic search for constrained interaction testing". In: *Empirical Software Engineering* 16 (2011), pp. 61–102.

[8] Mustafa Al-Hajjaji et al. "IncLing: efficient product-line testing using incremental pairwise sampling". In: *ACM SIGPLAN Notices* 52.3 (2016), pp. 144–155.

[9] Christopher Henard et al. "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines". In: *IEEE Transactions on Software Engineering* 40.7 (2014), pp. 650–670.

[10] Christopher Henard et al. "Multi-objective test generation for software product lines". In: *Proceedings of the 17th International Software Product Line Conference*. 2013, pp. 62–71.

[11] Christopher Henard et al. "PLEDGE: a product line editor and test generation tool". In: *Proceedings of the 17th International Software Product Line Conference Co-Located Workshops*. 2013, pp. 126–129.

[12] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. "An algorithm for generating t-wise covering arrays from large feature models". In: *Proceedings of the 16th International Software Product Line Conference-Volume 1*. 2012, pp. 46–55.

[13] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. "Properties of realistic feature models make combinatorial testing of product lines feasible". In: *Model Driven Engineering Languages and Systems: 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings 14*. Springer. 2011, pp. 638–652.

[14] Sebastian Krieter et al. "YASA: yet another sampling algorithm". In: *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*. 2020, pp. 1–10.

[15] Chuan Luo et al. "Solving the t-wise Coverage Maximum Problem via Effective and Efficient Local Search-based Sampling". In: *ACM Transactions on Software Engineering and Methodology* 34.1 (2024), pp. 1–64.

[16] M. D. McIlroy. "Mass Produced Software Components". In: *NATO Conference on Software Engineering*. Garmisch, Germany, 1968.

[17] Flávio Medeiros et al. "A comparison of 10 sampling algorithms for configurable systems". In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 643–654.

[18] Marcilio Mendonca, Moises Branco, and Donald Cowan. "SPLOT: software product lines online tools". In: *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. 2009, pp. 761–762.

[19] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. "SAT-based analysis of feature models is easy". In: *Proceedings of the 13th International Software Product Line Conference*. 2009, pp. 231–240.

[20] Sebastian Oster, Florian Markert, and Philipp Ritter. "Automated incremental pairwise testing of software product lines". In: *International Conference on Software Product Lines*. Springer. 2010, pp. 196–210.

[21] Gilles Perrouin et al. "Pairwise testing for software product lines: comparison of two approaches". In: *Software Quality Journal* 20 (2012), pp. 605–643.

[22] Muhammad Sahak, Dayang NA Jawawi, and Shahliza A Halim. "An experiment of different similarity measures on test case prioritization for software product lines". In: *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 9.3-4 (2017), pp. 177–185.

[23] Siti Hawa Mohamed Shareef, Rabatul Aduni Sulaiman, et al. "A Systematic Literature Review of Test Case Prioritization Technique on Software Product Line Testing". In: *KSII Transactions on Internet and Information Systems (TIIS)* 18.10 (2024), pp. 2872–2894.

[24] R Aduni Sulaiman, Dayang NA Jawawi, and Shahliza Abdul Halim. "A dissimilarity with Dice-Jaro-Winkler test case prioritization approach for model-based testing in software product line". In: *KSII Transactions on Internet and Information Systems (TIIS)* 15.3 (2021), pp. 932–951.

[25] Shuai Wang et al. "Multi-objective test prioritization in software product line testing: an industrial case study". In: *Proceedings of the 18th International Software Product Line Conference-Volume 1*. 2014, pp. 32–41.

[26] Yi Xiang et al. "Looking for novelty in search-based software product line testing". In: *IEEE Transactions on Software Engineering* 48.7 (2021), pp. 2317–2338.