

Efficient Product-Line Testing using Cluster-Based Product Prioritization

Mustafa Al-Hajjaji*, Jacob Krüger*[†], Sandro Schulze*, Thomas Leich[†], and Gunter Saake*

*University of Magdeburg, Germany

Email: mustafa.alhajjaji, jacob.krueger, sandro.schulze, gunter.saake@ovgu.de

[†]Harz University of Applied Sciences, Germany

Email: tleich@hs-harz.de

Abstract—A software product-line comprises a set of products that share a common set of features. These features can be reused to customize a product to satisfy specific needs of certain customers or markets. As the number of possible products increases exponentially for new features, testing all products is infeasible. Existing testing approaches reduce their effort by restricting the number of products (sampling) and improve their effectiveness by considering the order of tests (prioritization). In this paper, we propose a cluster-based prioritization technique to sample similar products with respect to the feature selection. We evaluate our approach using feature models of different sizes and show that cluster-based prioritization can enhance the effectiveness of product-line testing.

I. INTRODUCTION

Software product-line engineering (SPLE) is an approach to develop a set of systems by considering their commonalities and differences in terms of features [5, 8, 23]. Thus, SPLE aims to facilitate systematic reuse, increase quality, decrease time to market, and reduce development and maintenance costs [5, 8]. Despite the aforementioned advantages, SPLE also poses new challenges with respect to software quality assurance.

Due to the exponential number of potential products that can be derived from a set of features, it is difficult to guarantee correctness of all features and their combinations. To reduce costs, several approaches have been proposed to test only a sample of these products, for instance with *combinatorial interaction testing* (CIT) [21, 22]. CIT systematically limits the number of products by covering a certain degree of feature interactions, for instance pairwise. While CIT limits the products to test, the sampled set can still contain many products for large product lines. For example, Johansen et al. [16] report that applying pairwise testing for the Linux kernel (6,888 features) requires 480 products with each comprising hundreds of test cases to be executed, thus, requiring huge efforts. As testing time is limited, testers aim at finding faults as fast as possible. Hence, approaches have been proposed to enhance the effectiveness of product-line testing by prioritizing the generated products [4, 15] or its test cases [19].

In this paper, we exploit the commonality of products in a product line. Our idea is to cluster products based on their feature selections to identify those that are syntactically similar. On these clusters, developers can apply different testing strategies: Firstly, they may select a subset of products from each cluster. Hence, the sample covers dissimilar products

and thus, most likely different feature interactions. Secondly, the developers may want to cover a specific cluster in more detail, for instance, because it contains commonly demanded products. Hence, test efforts can be reduced by enabling faster fault detection. Finally, clustering products can be useful to optimize the setup time for testing, such as in the automotive domain. For instance, instead of consuming time to change the testing infrastructure (e.g. changing a specific hardware) for each product, products can be clustered based on a specific parameter and the same testing setup can be used for the whole products in a cluster. We evaluate our approach using feature models of different sizes, including the Linux kernel with 6,888 features. We compare cluster-based prioritization to random orders and a heuristic technique (similarity-based prioritization [4]). The results for cluster-based prioritization show potential improvement in the effectiveness of product-line testing (i.e., increasing the early rate of fault detection).

More precisely, we contribute the following:

- We propose a cluster-based prioritization approach to cluster products. This allows us to sample products and prioritize them.
- We evaluate our approach compared to a heuristic technique (similarity-based prioritization [4]) and random orders.
- We assess the impact of having different cluster numbers.

The remaining paper is structured as follows: In Section II we introduce background for this article. We describe our approach in Section III and provide an extensive evaluation in Section IV. Then, we briefly describe related work within Section V and conclude in Section VI.

II. BACKGROUND

In this section, we provide an overview of *feature models* and *combinatorial interaction testing*.

A. Feature Model

A *feature model* (FM) is a hierarchical structure used to define the variability of product lines in terms of features (i.e., a user-visible increment of functionality). Moreover, feature diagrams are used for graphically representing features and the relations among them [17]. We display the diagram for a mobile-phone product line in Figure 1 to further explain feature dependencies.

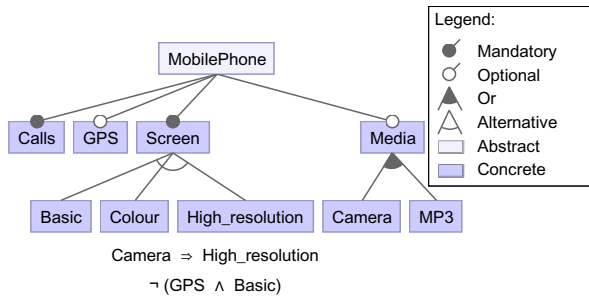


Fig. 1. Feature diagram of a *mobile-phone* product line.

Features themselves can either be mandatory or optional. A mandatory feature is required to appear in all generated products, such as feature *Calls* in Figure 1. In contrast, an optional feature is not required to be part of all products and enables customization.

Furthermore, features can be grouped into *alternative* and *or* dependencies. In an *alternative* group, only one feature can be selected in a particular configuration (e.g., only one of the features *Basic*, *Colour*, or *High_resolution* in Figure 1). Within *or* groups, at least one feature can be selected for a configuration. Additional dependencies, known as *cross-tree constraints*, describe relations between features (e.g., *requires* or *excludes* relationships). The combination of features is defined as a valid configuration if it satisfies the feature model dependencies. Each configuration can be used to generate a product implementing the selected features. In this paper, the terms configuration and product are used interchangeably.

B. Combinatorial Interaction Testing for Product Lines

Combinatorial interaction testing has shown its value in sampling test cases that cover parameter value combinations systematically [21, 22]. In the context of product-line testing, this approach is used to reveal faults that are caused by the interactions between features [22]. Using combinatorial interaction testing avoids exhaustive testing, which results from the combinatorial explosion problem [16]. In particular, the generated products cover all t -wise combinations of t features. For instance, in Table I, we list the valid configurations of the mobile-phone product line for $t = 2$ using the sampling algorithm ICPL [16]. Each combination of 2 features (pairwise) appears in at least one product. Khun et al. [18] report that applying pairwise testing can detect approximately 80% of faults. Although combinatorial interaction testing reduces the number of considered products significantly, this number can still be large [16].

Thus, several approaches have been proposed to prioritize products in order to find faults as fast as possible [4, 9, 26]. One of these approaches is similarity testing, which aims at increasing the interaction coverage [4, 15]. The results show that testing dissimilar products performs well at finding faults [4, 10, 15]. In this paper, we cluster similar products into groups, allowing to sample them based on their similarity or dissimilarity.

TABLE I
CONFIGURATIONS OF THE MOBILE-PHONE PRODUCT LINE IN FIGURE 1
CREATED WITH PAIRWISE SAMPLING.

ID	configurations
c_1	{Calls, Screen, Colour}
c_2	{Calls, GPS, Screen, High_resolution, Media, MP3}
c_3	{Calls, Screen, High_resolution, Media, Camera}
c_4	{Calls, Screen, Basic}
c_5	{Calls, Screen, High_resolution, Media, Camera, MP3}
c_6	{Calls, GPS, Screen, Colour, Media, MP3}
c_7	{Calls, GPS, Screen, High_resolution, Media, Camera}
c_8	{Calls, Screen, Basic, Media, MP3}
c_9	{Calls, GPS, Screen, High_resolution}

III. CLUSTER-BASED PRIORITIZATION

The main goal of our approach is to cluster products into subsets such that products in each set share common properties. As we illustrate in Figure 2, the input for our approach is a set of configurations. These configurations might be:

- All valid configurations of a product line.
- A set of configurations created with sampling algorithms.
- A set of configurations provided by domain experts.

In the following, we describe the two main steps, clustering and prioritization, of our approach, which we display in Figure 2. Clustering enables testers to cluster products into groups (e.g., cluster with the most demanded products). In addition, testers may wish to prioritize products within clusters based on certain criteria, such as the coverage, to find faults faster. For the latter, we use an existing approach to prioritize products, namely similarity-based prioritization [4].

A. Clustering

The commonality of configurations is measured with a clustering criterion. In this paper, we consider the similarity between products in terms of the selected features (their *configuration*, cf. Table I) as a criterion. The resulting clusters allow testers to select a sample of products from each or only a particular cluster. We consider the simple *K-means* algorithm to cluster products. While we argue that our clustering approach is independent from the used clustering algorithms, we plan to investigate whether using different cluster algorithms may influence the results.

B. Prioritization

In order to prioritize clustered products, we recognize two layers of prioritization:

- 1) *Intra-cluster* prioritization addresses the order of products in a cluster.
- 2) *Inter-cluster* prioritization addresses the order of clusters themselves.

In this paper, we only consider *intra-cluster* prioritization to prioritize products. Considering *inter-cluster* prioritization (i.e., from which clusters products are tested first) requires additional domain knowledge, for instance, which cluster contains more demanded products than others. Hence, during our evaluation we rely on the cluster ordering that is given by the clustering algorithms.

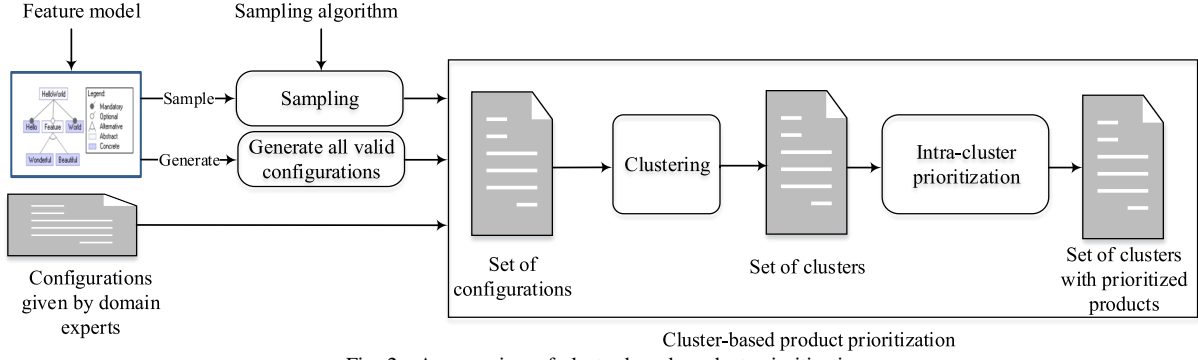


Fig. 2. An overview of cluster-based product prioritization.

To prioritize products in a cluster, we use similarity-based prioritization [4], that is, products are prioritized based on the similarity of their configurations. The product that is least similar to *all* previously tested ones is selected to be tested next. The goal of considering the similarity-based prioritization approach [4] is to increase the interaction coverage for products under test inside a particular cluster as soon as possible. Therefore, we first select the product with the maximum number of selected features. Considering our running example (cf. Table I), we have 3 configurations (c_2 , c_5 , c_7) with the maximum number of selected features (6 features). If multiple products have the same number of selected features, we pick one randomly. To explain the following steps, let us assume we have picked c_2 as the first product to be tested. Second, we select the product that is least similar to c_2 . The similarity between two products is measured using Hamming distance. Thus, the distance between two products c_i and c_j relative to the set of all features F is defined as follows:

$$distance(c_i, c_j, F) = 1 - \frac{|c_i \cap c_j| + |(F \setminus c_i) \cap (F \setminus c_j)|}{|F|} \quad (1)$$

$|c_i \cap c_j|$ is the number of selected features in the configurations c_i and c_j . $|(F \setminus c_i) \cap (F \setminus c_j)|$ is the number of deselected features in both configurations. The distance of two products ranges between 0 and 1, with similar products being close to 0. In contrast, values close to 1 indicate that the products are different.

Based on the values in Table I, we provide the following example to illustrate how the distances between configurations c_1 and c_2 are calculated: In c_1 , 3 features are selected while 6 are deselected. For c_2 in contrast, 6 features are selected while 3 are not. Hence, the value of $|c_i \cap c_j|$ is 2 and $|(F \setminus c_i) \cap (F \setminus c_j)|$ is 1. The distance between both configurations is then:

$$distance(c_1, c_2, F) = 1 - \frac{2+2}{9} = 0.556 \quad (2)$$

The distances between all configurations from Table I are listed in Table II. In our example, configurations c_1 , c_5 , and c_7 have the maximum distance (0.556) to c_2 . Similar to the previous step, if more than two products have the same distance, we select one randomly. For our example, we select c_1 as the second product to be tested.

Third, we select the next product to be tested that is least similar to *all* previously tested products. To calculate distances between more than two products, we use a strategy called maximum over distance minimum [4]. With this strategy, we first determine for all untested products their minimum distances to the set of already tested products. Then, we select the product that has the maximum of those minimum distances to be tested next. Considering again our example, we already selected two products c_2 and c_1 . In this step, to ensure we select the least similar product, we consider the minimum distances between the selected products and the other products (cf. Table II highlighted in bold). Then the product with the maximum distance of these minimum distances is selected. For our example, this applies to two products c_3 and c_8 . Again, one of them can be selected randomly. The resulting order of all products is $c_2, c_1, c_3, c_8, c_4, c_6, c_9, c_5$, and c_7 . We repeat the third step until all products are prioritized. In the next section, we evaluate the presented approach and discuss the results.

IV. EVALUATION

In this section, we formulate *research questions*, introduce our *subject systems*, and explain the *methodology* of our case study. Finally, we present and discuss the *results*.

A. Research Questions

For a given set of configurations, our approach aims to detect faults faster. To this end, we assess the effectiveness of cluster-based prioritization in terms of its fault detection rate compared to random orders and heuristic similarity-based prioritization [4]. To evaluate the impact of considering the *intra-cluster* prioritization, we compare it to the default order given by clustering algorithm. In particular, we answer the following questions:

- RQ-1** How does cluster-based prioritization perform compared to a heuristic similarity-based approach and random orders?
- RQ-2** How does *intra-cluster* prioritization influence the effectiveness of testing compared to using the default order provided by clustering algorithms?
- RQ-3** How does the number of clusters influence the effectiveness of testing?

To provide reasonable results, we consider several systems to answer our research questions.

TABLE II
DISTANCES BETWEEN THE 9 CONFIGURATIONS LISTED IN TABLE I

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
c_1	0	0.556	0.444	0.222	0.556	0.334	0.556	0.444	0.334
c_2	0.556	0	0.334	0.556	0.222	0.222	0.222	0.334	0.222
c_3	0.444	0.334	0	0.444	0.111	0.556	0.111	0.444	0.334
c_4	0.222	0.556	0.444	0	0.556	0.556	0.556	0.222	0.337
c_5	0.556	0.222	0.111	0.556	0	0.444	0.222	0.334	0.444
c_6	0.334	0.222	0.556	0.556	0.444	0	0.444	0.334	0.444
c_7	0.556	0.222	0.111	0.556	0.222	0.444	0	0.556	0.222
c_8	0.444	0.334	0.444	0.222	0.334	0.334	0.556	0	0.556
c_9	0.334	0.222	0.334	0.337	0.444	0.444	0.222	0.556	0

TABLE III
FEATURE MODELS USED FOR THE EVALUATION.

Feature Model	#Features	#Const.	CTCR	#Configs.
BattleofTanks	144	0	0%	459
FM_Test	168	46	28%	44
Printers	172	0	0%	181
BankingSoftware	176	4	2%	42
Electronic Shopping	290	21	11%	22
DMIS	366	192	93%	29
eCos 3.0 i386pc	1,245	2,478	99%	62
FreeBSD kernel 8.0.0	1,369	14,295	93%	77
Linux_2_6_28_6	6,888	6,847	99%	479
AFM5K	5,542	300	11%	685

#Const.: number of constants
 CTCR: cross-tree constraints representative
 #Configs.: number of configurations for pairwise sampling

B. Subject Systems

We consider a variety of subject systems from academia as well as real-world systems to evaluate our approach. Due to the lack of having open-source software product lines that have test cases and faults in the source code, we conduct our experiment using the feature models of the subject systems. The number of features in the subject systems ranges from 144 to 6,888, which allows us to evaluate the effectiveness of our approach even for large-scale systems, such as the Linux kernel. We selected these feature models because they have been used previously to evaluate the scalability of product-line testing [15, 16]. In Table III, we show the number of features, number of constraints (Const.), ratio of distinct features in cross-tree constraints to the number of features (CTCR), and number of valid configurations using pairwise sampling [3]. We created the configurations with the sampling algorithm ICPL [16].

C. Methodology

Given the feature models, we now explain our methodology for conducting the evaluation. In particular, we provide details about the faults to be detected, the clustering algorithms applied, and the metric used to assess the effectiveness of our approach.

a) Artificial Faults: Due to a lack of real-world product lines that have test cases and faults in the source code, we use simulated faults to evaluate our approach. For this, we applied a technique that has been used in previous studies on product-line testing [4, 10, 12, 26]: We randomly selected and marked features as containing faults. To simulate reality, the

faults are generated based on patterns of existing faults that were investigated in real-world product lines [1]. The simulated faults represent not only faults inside a single feature, but also interaction faults, which occur due to the interaction of features. In this paper, we assume that if the products contain these features and their combinations, the faults will be detected. Based on the reported patterns [1], we simulate faults up to 5-wise feature interactions. In addition, we assume that larger feature models exhibit potentially more feature interactions. Hence, the number of simulated faults on each feature model is $n/10$, where n is the number of features. That means that in our experiment the faults increase proportional with the size of a model.

b) Weka: For our evaluation we use the Waikato Environment for Knowledge Analysis (Weka) [13] version 3.8, an open source tool for machine learning and data mining. It provides several clustering algorithms, such as simple *K-means*, Hierarchical Cluster, or Em. In this paper, we use the simple *K-means* algorithm to cluster products, as it needs no deeper knowledge about clustering algorithms, thus, not biasing the results, and also can serve as a baseline for applying other algorithms in the future. To answer our third research question (RQ-3), we consider three values for K (i.e., different number of clusters): $K = 5$, $K = 10$, and $K = 15$.

c) Average Percentage of Faults Detected (APFD): We use the APFD metric developed by Elbaum et al. [11] to evaluate the effectiveness of fault detection. APFD is calculated by measuring the weighted average of faults detected for the system under test. The values range from 0 to 1 with higher values indicating faster fault detection rates. APFD is calculated as follows:

$$APFD = 1 - \frac{tf_1 + tf_2 + \dots + tf_m}{n * m} + \frac{1}{2n} \quad (3)$$

where n is the number of test cases, which represent products in our case, and m is the number of faults. Furthermore, tf_i is the position of the first test t that exposes the fault. We compute the APFD for all prioritization approaches. For the random order, we repeated the experiments 100 times to simulate an average performance. In Table IV, we report the average APFD values over 100 different sets of faults for our 10 subjected systems.

TABLE IV
AVERAGE APFD FOR CLUSTER-BASED PRIORITIZATION, RANDOM ORDERS, AND SIMILARITY-BASED PRIORITIZATION.

FM	APFD							
	Clustering						Random	Similarity
	K=5	K=5 (W/O ICP)	K=10	K=10 (W/O ICP)	K=15	K=15 (W/O ICP)		
BattleofTanks	0.700	0.697	0.699	0.699	0.703	0.698	0.689	0.708
FM_Test	0.740	0.718	0.741	0.741	0.740	0.723	0.660	0.738
Printers	0.760	0.751	0.760	0.760	0.760	0.754	0.745	0.749
BankingSoftware	0.585	0.551	0.587	0.587	0.561	0.554	0.536	0.608
Electronic Shopping	0.705	0.702	0.697	0.697	0.686	0.682	0.668	0.702
DMIS	0.716	0.676	0.699	0.673	0.670	0.630	0.639	0.733
eCos 3.0 i386pc	0.739	0.688	0.737	0.737	0.733	0.672	0.658	0.767
FreeBSD kernel 8.0.0	0.673	0.647	0.662	0.662	0.649	0.629	0.549	0.679
Linux_2_6_28_6	0.841	0.829	0.820	0.835	0.835	0.818	0.722	0.850
AFM5K	0.348	0.342	0.347	0.351	0.348	0.344	0.312	0.354
Average	0.681	0.660	0.677	0.663	0.669	0.650	0.618	0.689

K: number of clusters; W/O ICP: without considering intra-cluster prioritization

D. Results and Discussion

Regarding **RQ-1**, we compare cluster-based prioritization with $K = 5$, $K = 10$, and $K = 15$ (columns highlighted with gray in Table IV) to the heuristic similarity-based prioritization [4] and random orders with respect to the fault detection rate. Regardless the number of clusters, we observe that the average APFD values of cluster-based prioritization are higher than these of random orders for all feature models. In particular, the APFD values for cluster-based prioritization with $K = 5$, $K = 10$, and $K = 15$ are 0.681, 0.667, and 0.669, respectively, while the average APFD value of random order is 0.618. Hence, the improvements are 10.2%, 9.5%, and 8.3%.

If we look at the results for each feature model separately, we observe that for each one cluster-based prioritization is, on average, better than random orders. To support our observation, we apply the Mann-Whitney U test, a non-parametric statistical test, to investigate whether the differences to our approach are significant. From this test, we obtain a value, called p-value, representing the probability that two samples are equal. The significance level is 0.05, which means that a p-value less than or equal to 0.05 indicates significance. We observe that the difference between cluster-based prioritization and random orders is significant for all feature models, except *printers* and *BattleofTanks*, with p-values 0.30 and 0.34, respectively. All raw results of the experiment, including all p-values are publicly available¹.

Furthermore, we observe that the average APFD value of heuristic similarity-based prioritization (0.689) is slightly higher than the values for cluster-based prioritization. This applies especially with $K = 5$ with 0.681 for which the percentage of decrease is 1.2%.

Regarding **RQ-2**, we compare our approach to cluster-based prioritization without considering *intra-cluster* prioritization (i.e., taking the default order given by the clustering algorithm). We found that the average APFD values are higher when applying *intra-cluster* prioritization for $K = 5$, $K = 10$, and $K = 15$ with 0.681, 0.677, and 0.669, respectively than without

intra-cluster prioritization, where the values decrease to 0.660, 0.663, and 0.650. Hence, the percentages of improvement of considering *intra-cluster* prioritization are 3.18%, 2.1%, and 2.9%. Still, we notice that the improvement of prioritizing products within clusters is relatively small.

Regarding **RQ3**, we see that for our cluster-based prioritization fewer clusters result in higher APFD values. As we show in Table IV, the average APFD values for $K = 5$, $K = 10$, and $K = 15$ are 0.681, 0.677, and 0.669 respectively. The percentages improvement with $K = 5$ compared to $K = 10$ and $K = 15$ are 0.5% and 1.8%. The reason for the slight improvement is the impact of *intra-cluster* prioritization on the results. Fewer clusters result in more products within each cluster and, thus, the higher is the effect of *intra-cluster* prioritization. The following supports the aforementioned reasoning: Without considering *intra-cluster* prioritization the results are varying. For instance, we observe that the average APFD values for $K = 5$, $K = 10$, and $K = 15$ are 0.660, 0.663, and 0.650 respectively. In particular, the percentages of improvement of $K = 10$ compared to $K = 5$ and $K = 15$ are 0.4%, and 2.0%. We conclude that less clusters improve the results if *intra-cluster* prioritization is considered.

To summarize our findings, we answer our research questions as follows:

- RQ-1** Cluster-based prioritization on average performs better than random orders but slightly worse than heuristic similarity-based prioritization. However, cluster-based prioritization enables testers to select subsets of all products (e.g., select the cluster with the most demanded products), which cannot be done easily with heuristic prioritization, as it requires to compare all products instead of clusters.
- RQ-2** Considering the default order of clusters is slightly worse than prioritizing products overall or in a cluster. Still, clustering provides comparable results and further investigations seem promising.
- RQ-3** A higher number of clusters decreases the APFD value on average. Hence, increasing the number of clusters

¹http://www.witi.cs.uni-magdeburg.de/iti_db/research/spl-testing/CP/

influences badly on the testing effectiveness.

Still, clustering with and without prioritization achieves similar and occasionally better results compared to the heuristic approach, for instance for the *Printers* product line and $K = 15$, as we show in Table IV. We plan to further investigate why clustering performs better in such cases and whether we can improve its performance based on our findings.

E. Threats to Validity

In the following, we discuss the internal and external threats to validity that may affect our results.

There is a potential threat to internal validity related to the random distribution of the seeded faults. To mitigate this threat, we generated 100 sets of faults for each feature model. In each set, we selected 10% of each model's features and marked them as faulty. We argue that the random distribution of the seeded faults is better than building on non-representative distribution. Another internal threat is that we compared our approach to random orders. To mitigate random effects, we repeated those experiments 100 times.

A potential external validity threat related to the nature of feature models is that cluster-based prioritization may not provide similar results for different feature models. To alleviate this threat, we considered different feature model sizes with different complexities, including a version of the Linux kernel with 6,888 features. Furthermore, the selected clustering algorithm may influence the results. To alleviate this threat, we plan to consider other algorithms in future work.

V. RELATED WORK

A. Product Prioritization

Several approaches have been proposed to prioritize products based on different criteria. Using common feature model metrics, Sánchez et al. [26] propose five prioritization criteria. They compare their effectiveness and observe that different orderings of the same product line may lead to a significant difference in the rate of early fault detection. Complementary to their criteria, we propose cluster-based approach to prioritize products. Henard et al. [15] sample and prioritize products at the same time. They employ a search-based approach to generate products based on similarity among them. With our cluster-based approach, we focus on prioritizing products to exploit the similarity among them, which is compatible with any sampling technique. In previous work [4], we propose a heuristic similarity-based prioritization approach to prioritize products based on the similarity between them. In addition, we propose a heuristic approach that considers the similarity between products during the sampling process [2]. In this paper, we combine cluster-based prioritization to the aforementioned approach in order to enhance the product-line testing effectiveness.

Lity et al. [20] adopt graph algorithms to optimize product orders in order to reduce the incremental product-line analysis efforts. Combining their approach with cluster-based prioritization may reduce the effort in regression analysis. Baller et al. [6] introduce a framework to prioritize products under test

based on the selection of adequate test suites with regard to cost and profit objectives. The limitation of this approach is that it requires all products and their relation to test cases and test goals in advance. To tackle this limitation, Baller et al. in [7] propose an incremental test suite optimization approach for product-line testing that uses a symbolic representation in terms of feature constraints. However, further experiments are required to evaluate the effectiveness of their approach. In contrast, we use the similarity among configurations as criteria to prioritize them with clustering algorithms.

Sánchez et al. [25] prioritize products based on their non-functional attributes, such as feature size and the number of changes in a feature. These information are often not provided, especially in black-box testing. Devroey et al. [9] perform statistical analysis of a usage model in order to select the products with high probability to be executed. Both approaches are complementary to ours and apply different ideas.

B. Test Case Prioritization

In product-line testing, test case prioritization is used to reschedule test case execution to increase the rate of fault detection. Lachmann et al. in [19] propose an integration testing approach based on structural and behavioral deltas with a dissimilarity approach to prioritize test cases. Since these approaches mainly focus on test cases prioritization, combining them with cluster-based prioritization may enhance the effectiveness of product-line testing.

In single system testing, as surveyed by Yoo et al. in [27], efforts have been made to prioritize test cases. For instance, Rothermel et al. [24] describe several techniques for using test execution information to prioritize test cases in regression testing. This information includes code coverage and an estimation on the ability of test cases to detect faults. Hemmati et al. [14] report that selecting dissimilar test cases outperforms selecting test cases based on a coverage criterion. In contrast to these works, we propose cluster-based product prioritization to prioritize products in a product line, which could also be considered for test cases.

In the context of product lines, it might be that some of these approaches can be applied to prioritize either the products or the test cases. For example, Yoo et al. [28] prioritize test cases by classifying them into clusters. Then, they prioritize the clusters by utilizing domain expert judgment. We follow a similar approach but for products and do not consider domain expert judgment, which we may utilize in future work.

VI. CONCLUSIONS

The increasing interest in variable software systems in academia and industry requires adopted types of testing techniques to improve their quality. Most challenging in product-line testing is the large number of possible products that can be generated. While the number of products can already be reduced using combinatorial interaction testing, testers wish to find faults as fast as possible due to the limited testing time.

In this paper, we propose cluster-based product prioritization to cluster products into different subsets such that products in

each group share common features. This allows to sample these products and, thus, reduce the costs of testing. We evaluate our approach using different sizes of feature models. The results show that, on average, our approach performs slightly worse than current techniques but better than random orders. As a result, we argue that further investigations are necessary to improve cluster-based product prioritization.

In future work, we plan to consider additional clustering algorithms and different similarity measures. Furthermore, we aim to consider more product lines.

ACKNOWLEDGMENTS

We thank Sebastian Krieter for interesting discussions about the approach. This research is supported by DFG grants LE 3382/2-1 and SA 465/49-1.

REFERENCES

- [1] I. Abal, C. Brabrand, and A. Wasowski. 42 Variability Bugs in the Linux Kernel: A Qualitative Analysis. In *ASE*, pages 421–432. ACM, 2014.
- [2] M. Al-Hajjaji, S. Krieter, T. Thüm, M. Lochau, and G. Saake. IncLing: Efficient Product-Line Testing Using Incremental Pairwise Sampling. In *GPCE*, pages 144–155. ACM, 2016.
- [3] M. Al-Hajjaji, J. Meinicke, S. Krieter, R. Schröter, T. Thüm, T. Leich, and G. Saake. Tool Demo: Testing Configurable Systems with FeatureIDE. In *GPCE*, pages 173–177. ACM, 2016.
- [4] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake. Effective Product-Line Testing Using Similarity-Based Product Prioritization. *SoSyM*, pages 1–23, 2016.
- [5] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.
- [6] H. Baller, S. Lity, M. Lochau, and I. Schaefer. Multi-Objective Test Suite Optimization for Incremental Product Family Testing. In *ICST*, pages 303–312. IEEE, 2014.
- [7] H. Baller and M. Lochau. Towards Incremental Test Suite Optimization for Software Product Lines. In *FOSD*, pages 30–36. ACM, 2014.
- [8] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.
- [9] X. Devroey, G. Perrouin, M. Cordy, P.-Y. Schobbens, A. Legay, and P. Heymans. Towards Statistical Prioritization for Software Product Lines Testing. In *VaMoS*, pages 10:1–10:7. ACM, 2014.
- [10] X. Devroey, G. Perrouin, A. Legay, P.-Y. Schobbens, and P. Heymans. Search-based Similarity-Driven Behavioural SPL Testing. In *VaMoS*, pages 89–96. ACM, 2016.
- [11] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing Test Cases for Regression Testing. *SEN*, 25(5):102–112, 2000.
- [12] F. Ensan, E. Bagheri, and D. Gasevic. Evolutionary Search-Based Test Generation for Software Product Line Feature Models. In *CAiSE*, volume 7328, pages 613–628. Springer, 2012.
- [13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
- [14] H. Hemmati and L. Briand. An Industrial Investigation of Similarity Measures for Model-Based Test Case Selection. In *ISSRE*, pages 141–150. IEEE, 2010.
- [15] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Le Traon. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines. *TSE*, 40(7):650–670, 2014.
- [16] M. F. Johansen, Ø. Haugen, and F. Fleurey. An Algorithm for Generating T-Wise Covering Arrays from Large Feature Models. In *SPLC*, pages 46–55. ACM, 2012.
- [17] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, 1990.
- [18] D. R. Kuhn, D. R. Wallace, and A. M. Gallo Jr. Software Fault Interactions and Implications for Software Testing. *TSE*, 30(6):418–421, 2004.
- [19] R. Lachmann, S. Lity, F. E. Fürchtegott, M. Al-Hajjaji, and I. Schaefer. Fine-Grained Test Case Prioritization for Integration Testing of Delta-Oriented Software Product Lines. In *FOSD*, pages 1–10. ACM, 2016.
- [20] S. Lity, M. Al-Hajjaji, T. Thüm, and I. Schaefer. Optimizing Product Orders Using Graph Algorithms for Improving Incremental Product-Line Analysis. In *VaMoS*, pages 60–67. ACM, 2017.
- [21] S. Oster, F. Markert, and P. Ritter. Automated Incremental Pairwise Testing of Software Product Lines. In *SPLC*, pages 196–210. Springer, 2010.
- [22] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. Le Traon. Automated and Scalable T-Wise Test Case Generation Strategies for Software Product Lines. In *ICST*, pages 459–468. IEEE, 2010.
- [23] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
- [24] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Prioritizing Test Cases for Regression Testing. *TSE*, 27(10):929–948, 2001.
- [25] A. Sánchez, S. Segura, J. Parejo, and A. Ruiz-Cortés. Variability Testing in the Wild: The Drupal Case Study. *SoSyM*, pages 1–22, 2015.
- [26] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. A Comparison of Test Case Prioritization Criteria for Software Product Lines. In *ICST*, pages 41–50. IEEE, 2014.
- [27] S. Yoo and M. Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. *STVR*, 22(2):67–120, 2012.
- [28] S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering Test Cases to Achieve Effective and Scalable Prioritisation Incorporating Expert Knowledge. In *ISSTA*, pages 201–212. ACM, 2009.