Delta-Oriented Product Prioritization for Similarity-Based Product-Line Testing

Mustafa Al-Hajjaji*, Sascha Lity†, Remo Lachmann†, Thomas Thüm† Ina Schaefer† and Gunter Saake*
*University of Magdeburg, Germany
†TU Braunschweig, Germany

***Abstract*—Testing every product of a software product line (SPL) is often not feasible due to the exponential number of products in the number of features. Thus, the order in which products are tested matters, because it can increase the early rate of fault detection. Several approaches have been proposed to prioritize products based on configuration similarity. However, current approaches are oblivious to solution-space differences among products, because they consider only problem-space information. With delta modeling, we incorporate solution-space information in product prioritization to improve the effectiveness of SPL testing. Deltas capture the differences between products facilitating the reasoning about product similarity. As a result, we select the most dissimilar product to the previously tested ones, in terms of deltas, to be tested next. We evaluate the effectiveness of our approach using an SPL from the automotive domain showing an improvement in the effectiveness of SPL testing.**

I. Introduction

Software product-line engineering (SPLE) is a technique empowering software companies to compose software systems that meet the requirements of individual customers. Using SPLE, software systems can be developed efficiently by considering their commonalities and differences in terms of features to facilitate systematic reuse [4]. A feature is defined as an increment to the functionality that is recognized by customers [6]. The set of software systems sharing a set of features, but also containing some degree of variability are defined as a software product line (SPL). SPLs aim to reduce development and maintenance costs, to increase quality, and to decrease time to market. Several companies such as Hewlett-Packard, Toshiba, and General Motors have adapted their software development process to SPLE [4]. Despite the aforementioned potential advantages of SPLE, it poses new challenges for quality assurance.

Testing is a necessary step in software development processes. It consumes typically more than 50% of the whole development costs [12]. While testing a single software system is already difficult, testing SPLs is even more challenging due to the typically exponential number of products in the number of features. Thus, several approaches have been proposed to reduce the number of products to be tested, such as combinatorial interaction testing (CIT) techniques [15, 23].

Although the number of products can be reduced significantly using CIT techniques, the number of products to be tested can be still large. For example, Johansen et al. report that the number of generated products for the Linux kernel with $6,888$ features using pairwise CIT is still 488 [15]. In pairwise testing, each combination of two features is required to appear at least in one product. Still, testing the reduced number of products in practice with limited testing resources is difficult. Hence, testers wish to find faults faster by prioritizing products under test w.r.t. their likelihood of detecting new faults.

Several approaches have employed prioritization in SPL testing to increase the effectiveness of testing [3, 5, 9, 14]. These approaches focus mainly on prioritizing products based on configurations (i.e., selection of features only). The idea of similarity-based prioritization is to select a configuration to be tested that is the least similar to the previously tested configurations. However, configuration prioritization does not represent all actual differences between products. For instance, some features have more impact on solution-space artifacts than others. Hence, considering solution-space information for prioritization could improve the effectiveness of SPL testing.

In this paper, we propose delta-oriented product prioritization. This allows us to reason about differences between solution-space artifacts, without generating and comparing the complete products. Using delta-modeling artifacts [8] to prioritize products provide more detailed information about products than the feature selection, which may increase the rate of early fault detection. In case testers wish to consider configuration prioritization in product prioritization, we provide a combined approach of delta-oriented and configuration prioritization, where their weights can be adjusted using weighting factors. We investigate the effectiveness of our product prioritization approach compared to a default order of a sampling algorithm and random orders using an automotive SPL. The corresponding results show a potential improvement in the effectiveness of SPL testing in terms of the fault detection rate. In particular, we make the following contributions: (1) We propose an approach for product prioritization which uses delta modeling [8] for similarity-based prioritization avoiding the generation of complete products on solution-space level. (2) We provide a combined approach of configuration and delta-oriented prioritization in similarity-based product prioritization.

II. Background

A. Feature Modeling and Combinatorial Interaction Testing

A feature model is a hierarchical structure (cf. Figure 1) used to define and capture a set of features $F_{SPL} = \{f_1, \ldots, f_n\}$ and their dependencies [16]. A combination of a set of features, which satisfies the dependencies among them, is called a valid configuration F_p specifying a product $p \in P_{SPL}$. By P_{SPL}, we

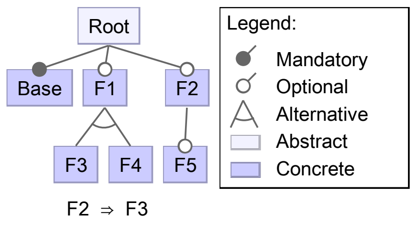


Fig. 1. A sample feature diagram

refer to set of all products defined by corresponding valid feature configurations.

Besides the definition of the configuration space, feature models are mainly used by sampling algorithms to derive representative subsets of configurations. Sampling algorithms (e.g., MoSo-PoLiTe [23]) apply CIT to reduce the number of generated products to be tested while achieving a certain degree of coverage [15, 23]. In the following, we give an overview on the MoSo-PoLiTe sampling algorithm, which will be used to generate a subset of products in our evaluation.

MoSo-PoLiTe [23] generates a minimal set of products covering all pairwise combinations. It works as follows: the valid combinations of pair features, w.r.t. the feature model, are generated. Then, each time the algorithm creates a valid configuration, it starts with the first combination and iteratively adds, if possible, the remaining combinations. The algorithm continues the previous step until all pairwise feature combinations are covered by at least one configuration. Typically, MoSo-PoLiTe outputs an ordered list of products, which will be used in our evaluation. In this paper, we consider the sampling algorithm MoSo-PoLiTe to investigate whether it has an effective order as it is the main algorithm used to sample the used subject SPL. In future work, we plan to consider other sampling algorithms. While sampling algorithms restrict the number of generated products, the number of products can still be large considering the limited testing time in practice. Hence, testers try to find faults as soon as possible by prioritizing products.

Several approaches have been proposed to prioritize products [3, 14, 25]. However, these approaches consider only problem-space information (i.e., the feature selection) to differentiate between products. Considering only problem-space information to differentiate between products does not represent all the actual differences among products. Hence, including solution-space information in product prioritization may enhance the SPL testing effectiveness.

B. Delta-Oriented Software Product Lines

Delta modeling has been proposed as an artifact-independent and modular variability modeling technique to develop SPLs [8]. Based on a preselected *core product* $p_{core} \in P_{SPL}$, differences are specified to transform the *core model* $m_{p_{core}}$ into product-specific models m_{p_i} for the remaining products $p_i \in P_{SPL}$ of the SPL under consideration. Each difference is defined by a change operation captured in a *delta* $\delta \in \Delta_{SPL}$. Thus, each delta δ comprises one particular operation, i.e., either an *addition* (add e) or a *removal* (remove e) of a model element e. By Δ_{SPL}, we refer to the set of all possible deltas for the current SPL. For each product $p_i \in P_{SPL}$, a delta set $\Delta_{p_i} \subseteq \Delta_{SPL}$

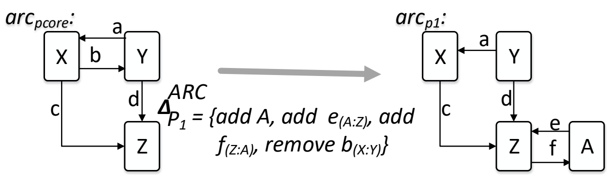


Fig. 2. The concept of delta-oriented software architectures

exists solely comprising the transformations to obtain the corresponding model m_{p_i} by applying all deltas $\delta \in \Delta_{p_i}$ subsequently to the core model. The product-specific delta sets are either predefined or can be determined based on application conditions, i.e., logical expressions using features $f \in F_{SPL}$ as variables, specified for each delta $\delta \in \Delta_{SPL}$. By giving a feature configuration F_p for product p, all application conditions are evaluated and those deltas are selected in the delta set Δ_p for which the evaluation results in `true`. The concept of delta modeling [8] is instantiated for various types of artifacts such as JAVA code [26], finite state machines [21], or architectures [20]. In this paper, we propose a prioritization technique independent from a concrete delta modeling instantiation. Nevertheless, we use delta-oriented architectures for illustration and evaluation.

Software architectures define the high-level structure of a software system [27] by describing the overall system *components* $C = \{c_1, \ldots, c_m\}$ and specifying the communication between them via *connectors* $CON = \{con_1, \ldots, con_l\}$ transmitting *signals* $\Pi = \{\pi_1, ..., \pi_k\}$. In the context of delta-oriented SPLs, for each product $p_i \in P_{SPL}$ a corresponding architecture model arc_{p_i} exists. Thus, based on a *core architecture model* $arc_{p_{core}}$, *architecture deltas* $\delta \in \Delta^{ARC}_{SPL}$ are defined and grouped in product-specific delta sets $\Delta^{ARC}_{p_i}$ to transform the core into an architecture model arc_{p_i}. The operations captured by a delta add/remove components, connectors, or signals.

Example 1: In Figure 2, a sample core architecture model $arc_{p_{core}}$ is shown on the left-hand side comprising the three components $C = \{X, Y, Z\}$ communicating via the four connectors $CON = \{a_{(Y:X)}, b_{(X:Y)}, c_{(X:Z)}, d_{(Y:Z)}\}$ by transmitting the four signals $\Pi = \{a, b, c, d\}$. To transform $arc_{p_{core}}$ into arc_{p_1} on the right-hand side, we apply the delta set $\Delta^{ARC}_{p_1}$, i.e., we remove one connector, add one component and two connectors.

Delta modeling [8] has been further adapted to improve SPL testing by reducing and prioritizing the number of executed test cases [17, 20, 21]. In this paper, we identify the commonalities and the differences between products by means of deltas to reason about their similarity on the solution-space level, e.g., between their product-specific architectures.

III. Similarity-Based Product Prioritization

Similarity-based testing is a technique used to select a subset of test cases, which aims to increase the early rate of fault detection [13]. Numerous approaches apply similarity-based testing to sample and prioritize products w.r.t. feature selections [3, 14]. In this paper, we prioritize products based on the similarity between them w.r.t. deltas. In this section, we present delta-oriented prioritization and the combined approach of configuration and delta-oriented prioritization.

A. Delta-Oriented Prioritization

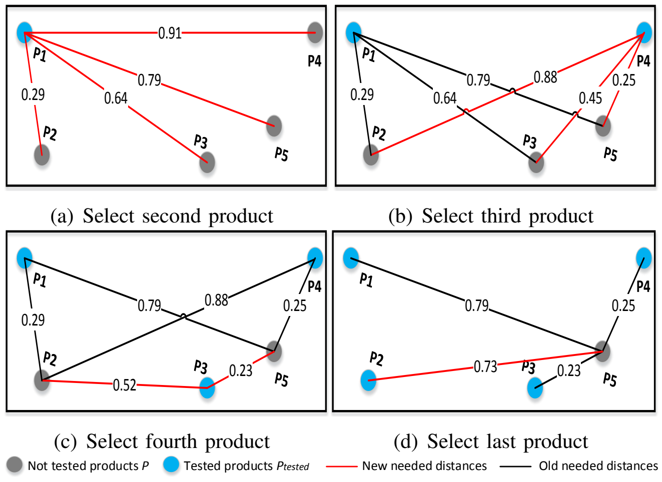
The input to our approach is a set of products which can be all valid products if the SPL is small, sample products that are generated using sampling algorithms if the SPL is large, or products given by domain experts. In addition to a product set, the delta model of the SPL is required.

With delta-oriented prioritization, we perform three steps to prioritize products by incorporating their differences by means of deltas to reason about their similarity. Therefore, (1) we select the first product which has the highest capability to detect the most faults at the beginning of the testing process, (2) we select the second product which is most dissimilar compared to the first product by means of deltas to be applied, and (3) the remaining products are incrementally selected by taking the delta-oriented similarity to all already tested products into account. In the following, we describe these steps in detail.

1) Choose First Product to Test: As the overall goal of our product prioritization is to detect faults as early as possible, we follow a strategy used for the analysis of the Linux kernel, where a specific product called *allyesconfig* is selected to be analyzed or tested first [11]. The *allyesconfig* is one of the largest products in the number of selected features. In contrast, we focus on the detection of faults in solution-space artifacts such as architectures and, therefore, reason about model size when selecting the first product to be tested. The first product builds the basis for the selection of the next products to be tested by incorporating their similarity to each other by means of model differences. By testing a large product first many faults that are caused by the selection of a single feature or the selection of two or more features can be detected [1].

In delta modeling [8], we observe a similar scenario, where the core product builds the basis for the specification of deltas to create the remaining product-specific models. In this paper, we combine both scenarios by (1) choosing a *complex core*, i.e., a certain product $p_{core} \in P_{SPL}$ with the largest set $ME_{p_{core}}$ of model elements comprised in the respective product-specific model $m_{p_{core}}$ and (2) selecting the core as first product to be tested. For architectures, the set ME_p of model elements is defined by the union of components C_p and connectors CON_p contained in the architecture arc_p. To find the complex core, we assume that either (1) the model size corresponds to the feature configuration size by means of selected features and, thus, the complex core has also the largest feature configuration, or (2) the developer/test engineer has this information based on domain and system knowledge. If there exist more than one product comprising the maximum number of elements, we select one of these products randomly as complex core for delta specification and first product to be tested. For testing a product, we apply standard testing techniques from single systems engineering, e.g., integration testing [7].

2) Choose Second Product to Test: After testing the core, we select the next product to be tested by incorporating the similarity compared to the core by means of model differences specified by deltas. By selecting the most dissimilar product, we facilitate the coverage of solution-space artifacts to be tested and, therefore, support early fault detection. To measure the similarity between products, we require a suitable distance metric and meaningful properties on which the chosen metric is applicable. Devroey et al. [10] investigated the impact of using different types of similarity measurements in test case prioritization such as Hamming, Jaccard, dice, anti-dice, and Levenshtein distance. They report that Hamming distance is one of the best distance functions. As we focus on solution-space artifacts such as architectures, we take the product-specific models and their differences by means of model elements into account to determine their similarity.



(a) Select second product (b) Select third product (c) Select fourth product (d) Select last product

Fig. 3. Delta-oriented prioritization with distance minimum

Following a naive approach, we have to create every product-specific model to determine the differences for the distance computation. Based on delta modeling [8], we already have the explicit knowledge about those differences specified by deltas and, thus, are able to determine the distance between two products by comparing their delta sets without generating each individual model. Each product-specific delta set comprises the particular deltas, i.e., change operations by means of additions and removals of elements, required for transforming the core into the corresponding model. The more deltas and, thus, change operations are common in both delta sets, the more similar are the respective products. In addition, the usage of delta modeling allows for a generalized definition of delta similarity. Therefore, the following generalized delta similarity function build the basis for the application of our approach in the context of the different delta modeling instantiation such as delta-oriented architectures.

The Hamming distance is a well-known distance measurement [13]. In order to handle product-specific delta sets for distance computation and, thus, for the comparison of two products p_i and p_j, we compare their corresponding delta sets Δ_{p_i} and Δ_{p_j}, and further the sets of deltas not applicable for both products. We define the delta similarity function $deltaDist_H$ based on the Hamming distance as follows

$$deltaDist_H(\Delta_{p_i}, \Delta_{p_j}, \Delta_{SPL}) = 1 - \frac{|\Delta_{p_i} \cap \Delta_{p_j}| + |(\Delta_{SPL} \setminus \Delta_{p_i}) \cap (\Delta_{SPL} \setminus \Delta_{p_j})|}{|\Delta_{SPL}|}, \quad (1)$$

TABLE I
FEATURE CONFIGURATIONS DERIVED BY USING PAIRWISE SAMPLING

ID	Configurations F_{p_i}
p_1	*{Root, Base, F1, F2, F3, ¬F4, F5}*
p_2	*{Root, Base, ¬F1, ¬F2, ¬F3, ¬F4, ¬F5}*
p_3	*{Root, Base, F1, F2, F3, ¬F4, ¬F5}*
p_4	*{Root, Base, F1, ¬F2, ¬F3, F4, ¬F5}*
p_5	*{Root, Base, F1, ¬F2, F3, ¬F4, ¬F5}*

where $|\Delta_{p_i} \cap \Delta_{p_j}|$ denotes the number of common deltas applicable for p_i as well as p_j, and $|(\Delta_{SPL} \setminus \Delta_{p_i}) \cap (\Delta_{SPL} \setminus \Delta_{p_j})|$ represents the number of common deltas which are not applicable to obtain their product-specific models. In the context of delta architectures, the Hamming distance function compare deltas specifying the addition and removal of components and connectors. The result ranges between 0 and 1, where a value close to 0 indicates that compared products have similar delta sets, whereas a value close to 1 indicates different products.

Example 2: Assume we want to test the five products $P_{SPL} = \{p_1, \ldots, p_5\}$ listed in Table I. We define p_1 as complex core selected to be tested first. Following an incremental process, we first determine the distances to the four untested products p_2, p_3, p_4, and p_5 as shown in Figure 3(a) using delta similarity function $deltaDist_H$. As a second step, we select the most dissimilar product to p_1 in terms of deltas. The distances between product p_1 and products p_2, p_3, p_4, and p_5 are 0.29, 0.64, 0.91, and 0.79, respectively. As p_4 has the largest distance (0.91) to p_1, we select p_4 as second product to be tested.

3) Choose Further Products to Test: For the selection of the third and further products to be tested from the set of remaining products $P_{SPL} \setminus P_{tested}$, we must take the similarity to all already tested products P_{tested} into account. By incorporating the distances to all previously tested products, the selection of the next product ensures the fast coverage of solution-space artifacts such as architectures and their elements and, thus, facilitate the early fault detection. Thus, we use a strategy called maximum over distance minimum [3] to calculate distances between more than two products. We perform two steps to determine the next product to be selected for testing. First, we determine for all untested products $P_{SPL} \setminus P_{tested}$ their minimum distances to the set of already tested products P_{tested}. Second, we determine the maximum of those minimum distances and select the corresponding product as next product to be tested. By incorporating the maximum of minimal distances, we are able to determine the best increment in artifact coverage and further are more stable against outliers which have a large distance to solely one already tested product and rather small distances to all other tested products. For the determination of the minimal distances, we define the relation $minDist \subseteq P_{SPL} \times P_{SPL}$ capturing the pair of untested and tested products for which the minimal distance exists such that for all $p \in P_{SPL} \setminus P_{tested}$ a corresponding pair exists as follows:

$$(p, p') \in minDist \Leftrightarrow \exists p' \in P_{tested} : \forall p'' \in P_{tested} \setminus \{p'\} : \quad (2)$$
$$deltaDist_H(\Delta_p, \Delta_{p'}, \Delta_{SPL}) \leq deltaDist_H(\Delta_p, \Delta_{p''}, \Delta_{SPL})$$

TABLE II
DISTANCES BETWEEN CONFIGURATIONS

	p_1	p_2	p_3	p_4	p_5
p_1	0	0.57	0.14	0.57	0.29
p_2	0.57	0	0.43	0.29	0.29
p_3	0.14	0.43	0	0.43	0.14
p_4	0.57	0.29	0.43	0	0.29
p_5	0.29	0.29	0.14	0.29	0

We use this relation for the product selection function $next_{min} : \mathscr{P}(P_{SPL}) \times \mathscr{P}(P_{SPL}) \rightarrow \mathscr{P}(P_{SPL})$ which is defined as follows:

$$next_{min}(P_{SPL}, P_{tested}) = \{p \in P_{SPL} \setminus P_{tested}, | \ \forall p' \in P_{SPL} \setminus P_{tested} :$$
$$\min_{p_i \in P_{tested}} deltaDist_H(\Delta_{p'}, \Delta_{p_i}, \Delta_{SPL}) > \quad (3)$$
$$\min_{p_j \in P_{tested}} deltaDist_H(\Delta_{p'}, \Delta_{p_j}, \Delta_{SPL})\}$$

If the result contains more than one selectable product, we select one of the potential candidates randomly. After testing the next product, we repeat the product selection until all products are tested or the provided testing resources are exhausted.

Example 3: Consider again our running example depicted in Figure 3. After selecting and testing product p_4 (cf. Example 2), we first update the distances between all untested and tested products shown in Figure 3(b). We select p_2 as next product to be tested, as the pair (p_3, p_4) has the maximum of minimum distances (0.45) compared to (p_2, p_1) (0.29) and (p_5, p_4) (0.25). The testing order results in p_1, p_4, p_3, p_2, and p_5.

B. Combining Configuration and Delta-Oriented Prioritization

In this section, we first show how products can be prioritized using configuration prioritization. Then, we present the combined approach of delta-oriented and configuration prioritization.

1) Configuration Prioritization: Configuration prioritization has been proposed to order products based on the similarity between them in terms of feature selection in previous work [3, 14]. The product that is least similar to the previous tested ones is selected to be tested next. Similar to the aforementioned steps in Section III-A, with configuration prioritization, we first select the product that has the maximum number of selected features to be tested. Second, we use the Hamming distance to measure similarity between products and we select the most dissimilar product to the first tested one. The Hamming distance between products w.r.t. the feature selection is computed as follows:

$$confDist(F_{p_i}, F_{p_j}, F_{SPL}) = \quad (4)$$
$$1 - \frac{|F_{p_i} \cap F_{p_j}| + |(F_{SPL} \ \setminus F_{p_i}) \cap (F_{SPL} \ \setminus F_{p_j})|}{|F_{SPL}|},$$

Third, we select the next product which is most dissimilar to *all* previously tested products. The third step is repeated until all products are tested or the testing resources are finished.

Example 4: Consider the configurations in Table I derived from the sample feature model shown in Figure 1 by applying a pairwise sampling algorithm. For each configuration F_{p_i}, the selected features $f \in F_{SPL}$ and deselected features $f' \in F_{SPL}$ are listed, where feature deselection are denoted by a ¬. Assume p_1 is the first product to be tested. The feature configuration distances are shown in Table II. By applying the configuration prioritization, we select p_2 as next most dissimilar product. In the end, we obtain p_1, p_2, p_4, p_5, and p_3 as the resulting testing order of products under test.

2) Combined Approach: We control the weight of each approach by introducing α as weighting factor. Hence, the value of α adjust the weight of the delta-oriented prioritization, whereas $1-\alpha$ represent the weight for the configuration prioritization as counterpart. The total combined distance for the product prioritization is defined as follows:

$$totalDist(F_{p_i}, F_{p_j}, F_{SPL}, \Delta_{p_i}, \Delta_{p_j}, \Delta_{SPL}) = \frac{\alpha \cdot deltaDist_{H/J} + (1-\alpha) \cdot confDist}{2} \quad (5)$$

Testers may adjust the α value based on the quality and the details of the given information, which can be used to differentiate between products in an SPL.

Example 5: Consider Example 4 and Example 3 again, where the testing orders are determined based on $\alpha = 0$ and $\alpha = 1$, respectively. Again, we use p_1 as complex core selected as first product under test. We use Equation 5 as well as the maximum of distance minimum to select the next products. For $\alpha = 0.0$, we obtain p_1, p_2, p_4, p_5, and p_3 as testing order. In contrast, for $\alpha = 1.0$, we derive p_1, p_4, p_3, p_2, and p_5 as testing order showing the impact of the variation of α.

IV. Evaluation

To assess our prioritization approach presented in Section III, we study the following research questions:

RQ 1 How does similarity-based prioritization using delta modeling *perform* compared to random orders and the default order of the sampling algorithm *MoSo-PoLiTe* [23]?

RQ 2 How does the combination of delta-oriented and configuration prioritization *influences* the results?

A. Subject Product Line

We evaluate our approach by means of an SPL from the automotive domain representing a Body Comfort System (BCS) [19]. The BCS has 27 mandatory and optional features, which comprise a total of 11,616 possible products. Applying pairwise sampling, the number of generated products is reduced to 17 products, where we add the core as product 18 [19]. For evaluation, we focus on the delta-oriented architecture specified for the BCS SPL [19]. The size of all the designed architectures ranges from 4 to 19 components. The average number of connectors in each architecture is 72 that transfer an average of 60 different signals between components.

B. Fault Injection

To measure the effectiveness of the proposed approach, we use seeded faults using a well-known method to assess the fault detection of a test suite [22]. We randomly select architecture model elements and mark them as containing faults. We assume that if the products contain these elements, the faults will be detected. Applying such a technique is common in the community [3, 10, 25] due to the lack of having access to case studies that have test cases and numerous real faults. Abal et al. report that some faults can be triggered because some features are *not* selected [1]. Thereupon, we consider the involved and the non-involved model elements when we seed faults. We consider two types of faults: single faulty elements to simulate faults in a single element and pairwise combinations of faulty elements to simulate faults caused by the interaction between elements. In the following steps, we show an example of how the pairwise faults are seeded.

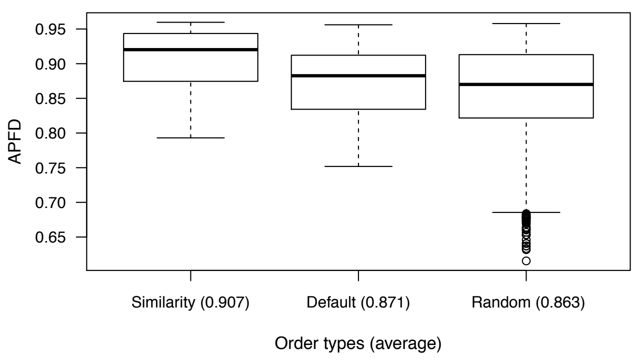


Fig. 4. The APFD distribution for similarity-based prioritization (Similarity), default order of sampling algorithm *MoSo-PoLiTe [23]* (Default), and random orders (Random)

At first, we select two elements (i.e., components) randomly. Assume that we select components c_3 and c_5. The decision of involving the component in a faulty combination or not ($\neg$) is decided randomly. In our case, assume that component c_3 is selected to be involved and component c_5 is selected to be not involved. The combination of the two components is $c_3 \wedge \neg c_5$. We ensure the validity of the generated combination w.r.t. the feature model and delta model. We generate 200 sets of faults (100 sets for single-wise interactions and 100 sets for pairwise interactions). In each set, we randomly select 10% of the faulty elements. We use average percentage of faults detected (APFD) as a metric to evaluate how fast faults are detected during testing. APFD values range from 0 to 1; higher values indicate faster fault detection. APFD is calculated by

$$\text{APFD} = 1 - \frac{te_1 + te_2 + \ldots + te_n}{n \cdot m} + \frac{1}{2n}, \quad (6)$$

where n is the number of test cases, which represent products in our case, m is the number of faults, and te_i is the position of the first test case t that exposes the fault e.

C. Results and Discussion

To show the effectiveness of our approach, we compare it against random orders as well as the default order of the sampling algorithm *MoSo-PoLiTe* [23], where an implicit order as part of its output is given by the positioning of the products within the output data structure. To answer **RQ 1**, we used delta-oriented prioritization to prioritize products. The boxplots in Figure 4 show the APFD distribution and the average for each approach over 200 sets of faults.

From *Figure* 4, it is obvious that delta-oriented prioritization (0.907) outperforms the random ordering (0.863) as well as the default order of *MoSo-PoLiTe* [23] sampling algorithm (0.871). To test whether the difference between our approach and the

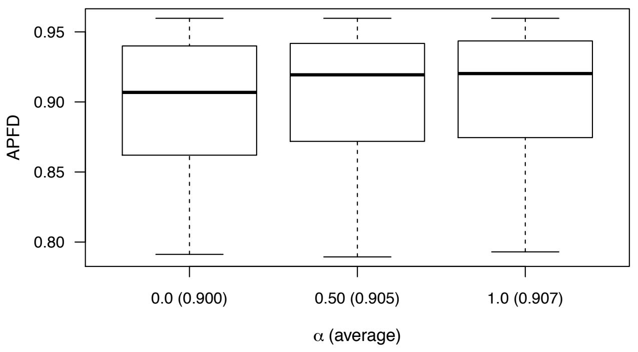


Fig. 5. The APFD distribution for similarity-based prioritization with considering different weights of delta-oriented prioritization represented by the value of α

default order of *MoSo-PoLiTe* as well as the random orders is significant, we use the Mann-Whitney U test. From the test, we obtain a probability value, called P-value. If the p-value is lower than 0.05, the difference is significant. Otherwise, the difference is not significant. In our results, we observe that the differences between our approach and the default order of *MoSo-PoLiTe* as well as the random orders are significant with p-value 0.0 for both approaches. Hence, regarding **RQ 1**, delta-oriented prioritization outperforms the random ordering as well as the default order of *MoSo-PoLiTe* [23].

Regarding **RQ 2**, in order to show whether the combination of configuration and delta-oriented prioritization influences the results, we consider three cases which represent different values of α. These different values adjust the weight of delta-oriented and configuration prioritization (*cf. Equation 5*). The three cases are: Case 1: α =0.0, $1-\alpha$= 1.0 (configuration prioritization). Case 2: α =0.50, $1-\alpha$= 0.50. Case 3: α =1.0, $1-\alpha$= 0.0 (delta-oriented prioritization). The proposed α values in this paper are for evaluation purposes. However, the α value can be adjusted by testers based on the quality of the provided information of SPLs.

Figure 5 shows the distribution and the average of APFD values, where the X-axis denotes different values of α, and the Y-axis denotes the APFD values. From *Figure* 5, we observe that including solution-space information to problem-space information improves the effectiveness of similarity-based prioritization. Furthermore, the results show that the average APFD value over 200 sets of faults when $\alpha = 0.0$ is 0.900, while it is 0.907 when $\alpha = 1.0$. Using the Mann-Whitney U test, we found that the difference between the orders of the generated products, where $\alpha = 0.0$ and $\alpha = 1.0$, is significant with p-value 0.04. We envision that adding more solution-space information may enhance the SPL testing effectiveness compared to configuration prioritization. However, adding problem-space information does not seem to give advantages.

D. Threats to Validity

An internal threat that may affect the results is the random distribution of the seeded faults. To mitigate this threat, we generated 200 sets of faults for the single-wise and pairwise interactions. In each set, we select 10% of architecture model elements and mark them as they are faulty. We assume that the random distribution of the seeded faults is better than building on non-representative distributions. Assuming 10% of the elements as seeded faults raises an internal threat that may affect the results. Hence, we conducted different experiments with other percentages that led to very similar results and, thus, to same interpretations. The raw results of all experiments are available[1]. Another internal threat is that we compared our approach to random orders. To mitigate the random effects, we repeated those experiments 100 times. Regarding external validity, we cannot ensure that our subject SPL is representative for real-world SPLs. The product line BCS already served as a benchmark to evaluate SPL testing techniques [17, 20, 21]. In addition, we are not aware of other industrial or even academic SPLs publicly available with their architecture models.

V. Related Work

In previous work [3], we proposed configuration prioritization to order products based on the similarity between them w.r.t. the feature selection. In a following work [2], we propose a heuristic incremental approach that considers the similarity between products during the sampling process. In this paper, we propose delta-oriented prioritization to order products based on the similarity between them w.r.t. deltas. In addition, we investigate the impact of having more information to differentiate between products. Henard et al. [14] sample products based on selected features. They employ a search-based approach to generate products based on similarity among them. In this paper, we exploit the commonalities and differences between products in terms of deltas to prioritize products. Devroey et al. [10] propose a search-based approach to select test cases for behavioral SPL models based on similarity. In our work, we consider similarity to prioritize products. Lity et al. [18] propose to reduce the incremental SPL analysis efforts by optimizing product orders using adopted graph algorithms. In their work, they select the next product that is the less different to the previous ones to be analyzed to reduce the effort in regression analysis. However, their work is based on an assumption that the analysis process will be finished in the given time. In our work, we select a product that is less similar to the previously tested ones to be tested next in order to detect faults as soon as possible.

Sánchez et al. [25] propose to prioritize products using common feature model metrics. We include information extracted from delta modeling in product prioritization. Devroey et al. [9] select the products with high probability to be executed using statistical analysis of an usage model. Parejo et al. [24] propose search-based approach that combines functional and non-functional attributes to optimize the product ordering. In our approach, we prioritize products based on their similarity in terms of deltas. Hence, our approach does not require expert knowledge to prioritize products.

Lachmann et al. [17] propose to prioritize test cases based on structural and behavioral deltas with a dissimilarity approach to

[1]http://wwwiti.cs.uni-magdeburg.de/iti_db/research/spl-testing/d/

prioritize message sequence chart test cases. In our work, we use these delta models to prioritize products. In single system testing, Yoo et al. [28] survey efforts that have been made to prioritize test cases in regression testing. Using some of these techniques in SPL testing to prioritize products or the test cases might enhance SPL testing.

VI. Conclusion and Future Work

In this paper, we propose similarity-based prioritization to order products based on their similarity w.r.t. deltas. In addition, we proposed a combined approach that considers configuration and delta-oriented prioritization in product prioritization. The results show that prioritizing products based on delta modeling can enhance the effectiveness of SPL testing. In future work, we plan to include more solution-space information, such as the source code to investigate whether it can improve the impact of similarity-based prioritization. In addition, we plan to measure how much time is required to achieve delta-oriented prioritization compared to configuration prioritization.

Acknowledgments

This work was partially funded by DFG under Priority Programme SPP1593: Design For Future–Managed Software Evolution.

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, T. Thüm, M. Lochau, J. Meinicke, and G. Saake. Effective Product-Line Testing Using Similarity-Based Product Prioritization. *SoSyM*, pages 1–23, 2016.

[4] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.

[5] 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.

[6] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *TSE*, 30(6):355–371, 2004.

[7] A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti. An approach to integration testing based on architectural descriptions. In *ICECCS*, pages 77–84. IEEE, 1997.

[8] D. Clarke, M. Helvensteijn, and I. Schaefer. Abstract Delta Modeling. In *GPCE*, pages 13–22. ACM, 2010.

[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] C. Dietrich, R. Tartler, W. Schröder-Preikschat, and D. Lohmann. Understanding Linux Feature Distribution. In *MISS*, pages 15–20. ACM, 2012.

[12] M. J. Harrold. Testing: A Roadmap. In *FSE*, pages 61–72. ACM, 2000.

[13] H. Hemmati, A. Arcuri, and L. Briand. Achieving Scalable Model-based Testing Through Test Case Diversity. *TOSEM*, 22(1):6:1–6:42, 2013.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer. Delta-Oriented Software Product Line Test Models-The Body Comfort System Case Study. Technical report, Technical report, TU Braunschweig, 2013.

[20] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz. Delta-Oriented Model-Based Integration Testing of Large-Scale Systems. *JSS*, 91:63–84, 2014.

[21] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity. Incremental Model-Based Testing of Delta-oriented Software Product Lines. In *TAP*, pages 67–82. Springer, 2012.

[22] A. P. Mathur. *Foundations of Software Testing*. Addison-Wesley Professional, 1st edition, 2008.

[23] S. Oster, F. Markert, and P. Ritter. Automated Incremental Pairwise Testing of Software Product Lines. In *SPLC*, pages 196–210. Springer, 2010.

[24] J. A. Parejo, A. B. Sánchez, S. Segura, A. Ruiz-Cortés, R. E. Lopez-Herrejon, and A. Egyed. Multi-objective test case prioritization in highly configurable systems: A case study. *JSS*, 122:287 – 310, 2016.

[25] 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.

[26] I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella. Delta-Oriented Programming of Software Product Lines. In *SPLC*, pages 77–91. Springer, 2010.

[27] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[28] S. Yoo and M. Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. *STVR*, 22(2):67–120, 2012.