# Scalable Sampling and Prioritization for Product-Line Testing

Mustafa Al-Hajjaji
University of Magdeburg
mustafa.al-hajjaji@st.ovgu.de

**Abstract:**
Exhaustively testing every product of a software product line (SPL) is a difficult task due to the combinatorial explosion of the number of products. Several sampling approaches have been proposed to select a subset of the products that can be used to test the SPL. However, these approaches do not scale to large SPLs such as the Linux kernel. Since the time budget of testing is usually limited, the order in which products are tested is important. Products have been prioritized based on domain knowledge or feature-model criteria, but not much attention has been paid to source code. We plan to use evolutionary testing approaches with different inputs to fitness functions to explore the configuration space of feature models. The configurations will be generated and prioritized automatically. Using criteria from feature model and source code, we want to investigate whether we can increase the efficiency of SPL testing. In this proposal, we present research questions that investigate how we can employ evolutionary testing approaches to increase the efficiency of SPL testing.

## 1 Introduction and Problem Statement

SPL engineering is an approach reusing a common set of features across a potentially large number of similar products systematically. Recently, SPLs are gaining acceptance and several companies such as Hewlett Packard, Toshiba, and General Motors adopt their software development process to SPLs [ABKS13]. SPL engineering addresses well-known needs of software engineering, such as reducing the cost of development and maintenance, increasing the quality, and decreasing the time to market [ABKS13]. Testing an SPL is a difficult task due to the amount of possible combinations between features which lead to explosion of possible products that need to be tested. The main concern is that it is not possible to test all the possible products because the resources for testing are usually limited. To tackle this problem, several approaches have been proposed to reduce the number of products to test, such as combinatorial interaction testing (CIT) [CDS07].

CIT is a promising approach that can perform the interaction testing between features in an SPL. T-wise interaction testing is a technique that has been used to detect faults which are caused by the interaction between features [JHF12] [OMR10]. Several sampling algorithms have been proposed to select a set of configurations to be tested [OMR10] [JHF12]. However, these algorithms still have a scalability problem of handling large feature models [LvRK+13]. Furthermore, these sampling approaches focus only on how to generate the minimum number of products for a given coverage. However, there are other parameters that may involve in the SPL testing, such as time and cost. Although CIT is an approach

to reduce the number of products to test, the number of the products can be too large, especially if the time budget is limited. Hence, in which order these products are tested is important in this context. The idea of prioritizing products is to find faults as soon as possible by starting testing products that are most likely to contain faults. The existing prioritization approaches consider only the domain knowledge [EBA+11] and feature model [AHTM+14]. As far as we know, there is no approach that considers source code for prioritization in SPL testing.

There are a few approaches combine sampling and prioritization [HPP+13] [EBG12]. These approaches use genetic algorithms to generate products based on certain objectives. In their approaches, they focus only on a few criteria based on feature models. We plan to investigate whether code-based criteria can be used to enhance the efficiency of SPL testing. We plan to propose evolutionary algorithms (EA) to explore the configuration space of feature models in order to generate the optimal products based on different objectives by using these criteria as inputs to the fitness functions of EA. Throughout this paper, the term *efficiency* represents the following: finding as many faults as possible with the least possible effort and increasing the rate of fault detection (i.e., a measure of how quickly faults are detected). In particular, we address the following research questions:

RQ1 How to increase the rate of fault detection for SPLs testing by prioritizing products for a given time budget?

RQ2 What are code-based metrics that enhance the efficiency of SPL testing?

RQ3 How to achieve efficient sampling (i.e., handling multiple parameters)?

## 2 Research Approach

To answer RQ1, we propose a similarity-based prioritization approach to prioritize configurations [AHTM+14]. This criterion measures the similarity between configurations. The motivation of using the similarity between configurations is that similar products are likely to contain the same faults. We evaluate the similarity between configurations using the Hamming distance. Hamming distance is mathematically given by the following formula, where $c_i$ and $c_j$ are configurations and $F$ is the set of all features in an SPL.

$$d(c_i, c_j, F) = 1 - \frac{|c_i \cap c_j| + |(F \backslash c_i) \cap (F \backslash c_j)|}{|F|}$$

At first, we select the first configuration with the maximum number of selected features because it covers most faults in individual features and enables selection of the next configuration with large distance. Then, we calculate the distance between this configuration and all the other selected configurations and we select the one with the maximum distance. We continue the process until all the configurations are ordered. Comparing to the random orders and the default order of sampling algorithms which are used to sample configurations from feature models, similarity-based prioritization approach showed promising results on different size of SPLs with respect to increasing the rate of fault detection [AHTM+14].

In Section 2.1, we present how we can address RQ2. To answer RQ3, we plan to employ EA to search the possible configuration space of feature models. The fitness function will

be used to capture SPL testing parameters (objectives) that maximize the efficiency of SPL testing. EA are generic, in a sense that different fitness functions can be defined to achieve different goals. Hence, we can apply the same EA to different SPL testing scenarios. In our work, we plan to use cuckoo search (CS) algorithm. CS optimization algorithm is a recently proposed EA, based on the behavior of certain species of the cuckoo birds such as the *guira* and the *ani*, combined with the displacement mode of many animals and insects such as sharks and some bird species. This movement pattern has been described as a heavy-tailed probability distribution and called the Lévy Flight pattern [YD09]. See our earlier technical report [AH14] for more details about the CS algorithm and other related issues of SPL testing.

### 2.1 Code-Based Criteria for SPLs

Regarding RQ2, we present examples of code-based criteria that we plan to use in our project. The goal of using these criteria is to measure the complexity of products and give higher priority to the products with higher complexity. We plan to investigate whether these criteria and others can increase the efficiency of SPL testing.

- **Scattering Degree (SD) and Tangling Degree (TD)** The SD metric is the number of the occurrences of a feature in other feature's expressions. This metric is measured by extracting feature names from feature expressions and calculate the average number of times the feature appears in the other feature's expressions.

  TD metric is the number of different features that occur in a feature expression. A lower TD is better, because high TD of features in feature expressions may impair program comprehension [LAL+10].

- **Average Nesting Depth of features (AND)** The AND metric represent the average nesting depth of features. It can be measured by calculating the average nesting depth of features for each product. Since we extract nested features form feature expressions, this metric is useful for discussions on program comprehension [LAL+10].

### 2.2 Research Methodology

This thesis will rely on the combination of a systematic review, experiments, and industrial case studies. We use systematic reviews to synthesize the research results, to summarize the existing evidence for SPL testing, and to aid in the identification of gaps in the current research. A main portion of this research proposal is about using search-based techniques for SPL testing. This thesis will be approached through experimental research methods. We will implement and evaluate our approaches in FeatureIDE [TKB+14]. We plan to apply our approaches using case study with the industrial use of FeatureIDE in order to evaluate our approaches with real-world product lines.

## 3 Conclusion

We believe that there is a lack of research using code-based criteria to prioritize products. In addition to feature-model criteria, we want to investigate what code-based metrics are

that may increase the efficiency of SPL testing. We plan to employ evolutionary algorithms by incorporating these criteria into fitness functions.

## References


[ABKS13] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.

[AH14] Mustafa Al-Hajjaji. Scalable and Efficient Sampling for Product-Line Testing. Technical Report FIN-003-2014, University of Magdeburg, Germany, October 2014.

[AHTM+14] Mustafa Al-Hajjaji, Thomas Thüm, Jens Meinicke, Malte Lochau, and Gunter Saake. Similarity-Based Prioritization in Software Product-Line Testing. In *SPLC*, pages 197–206, New York, NY, USA, 2014. ACM.

[CDS07] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Interaction Testing of Highly-Configurable Systems in the Presence of Constraints. In *ISSTA*, pages 129–139. ACM, 2007.

[EBA+11] Alireza Ensan, Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Yevgen Biletskiy. Goal-Oriented Test Case Selection and Prioritization for Product Line Feature Models. In *ITNG*, pages 291–298. IEEE, 2011.

[EBG12] Faezeh Ensan, Ebrahim Bagheri, and Dragan Gasevic. Evolutionary Search-Based Test Generation for Software Product Line Feature Models. In *CAiSE*, volume 7328, pages 613–628. Springer, 2012.

[HPP+13] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Multi-Objective Test Generation for Software Product Lines. In *SPLC*, pages 62–71, New York, NY, USA, 2013. ACM.

[JHF12] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. An Algorithm for Generating T-Wise Covering Arrays from Large Feature Models. In *SPLC*, pages 46–55, NY, USA, 2012. ACM.

[LAL+10] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In *ICSE*, pages 105–114, Washington, DC, USA, May 2010. IEEE.

[LvRK+13] Jörg Liebig, Alexander von Rhein, Christian Kästner, Sven Apel, Jens Dörre, and Christian Lengauer. Scalable Analysis of Variable Software. In *ESECFSE*, pages 81–91, New York, NY, USA, August 2013. ACM.

[OMR10] Sebastian Oster, Florian Markert, and Philipp Ritter. Automated Incremental Pairwise Testing of Software Product Lines. In *SPLC*, pages 196–210, Berlin, Heidelberg, 2010. Springer.

[TKB+14] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *SCP*, 79(0):70–85, January 2014.

[YD09] Xin-She Yang and S. Deb. Cuckoo Search via Levy Flights. In *NaBIC*, pages 210–214. IEEE, 2009.