Multi-objective Test Generation for Software Product Lines

Christopher Henard
SnT, University of Luxembourg
Luxembourg, Luxembourg
christopher.henard@uni.lu

Mike Papadakis
SnT, University of Luxembourg
Luxembourg, Luxembourg
michail.papadakis@uni.lu

Gilles Perrouin*
PReCISE, University of Namur
Namur, Belgium
gilles.perrouin@fundp.ac.be

Jacques Klein
SnT, University of Luxembourg
Luxembourg, Luxembourg
jacques.klein@uni.lu

Yves Le Traon
SnT, University of Luxembourg
Luxembourg, Luxembourg
yves.letraon@uni.lu

ABSTRACT

Software Products Lines (SPLs) are families of products sharing common assets representing code or functionalities of a software product. These assets are represented as features, usually organized into Feature Models (FMs) from which the user can configure software products. Generally, few features are sufficient to allow configuring millions of software products. As a result, selecting the products matching given testing objectives is a difficult problem.

The testing process usually involves multiple and potentially conflicting testing objectives to fulfill, e.g. maximizing the number of optional features to test while at the same time both minimizing the number of products and minimizing the cost of testing them. However, most approaches for generating products usually target a single objective, like testing the maximum amount of feature interactions. While focusing on one objective may be sufficient in certain cases, this practice does not reflect real-life testing situations.

The present paper proposes a genetic algorithm to handle multiple conflicting objectives in test generation for SPLs. Experiments conducted on FMs of different sizes demonstrate the effectiveness, feasibility and practicality of the introduced approach.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Algorithms, Experimentation

Keywords

Software Product Lines, Test Generation, Multi-objective Optimization, Genetic Algorithms, Feature Models

*FNRS Postdoctoral Researcher.

SPLC 2013, August 26 - 30 2013, Tokyo, Japan

1. INTRODUCTION

The software industry is increasingly building software families consisting of similar systems with many variations [5], called variants. Such families are known as Software Product Lines (SPLs). A Software Product Line (SPL) has been defined as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [3]. Software Product Line Engineering (SPLE) [27] is a software development paradigm designed to handle these variants. This approach results in many benefits, like reducing the maintenance effort, the software development costs and the time of products to market [7]. The features of a SPL are usually organized into a Feature Model (FM) [17] from which the user can configure and derive [28] tailored software products.

Testing a SPL is a inherent difficult task [21]. Indeed, only few features are sufficient to enable the configuration of thousands or millions of variants. For instance, a FM of a video player with 71 features [22] allows configuring more than 1.65×10^{13} different products. Therefore, testing all the products of a SPL with an acceptable cost is almost unmanageable. To overcome this problem, approaches have been proposed to reduce the number of products to test [8]. For instance, Combinatorial Interaction Testing (CIT) [4] has been identified as a relevant approach to sample the configuration space. This technique is based on the observation that most of the faults are due to the interaction between a small number of features. As a result, only the products exercising all the interactions between any t features (t-wise) should be tested. This approach has been adapted to SPL testing, e.g. [15, 25].

In real-life situations, selecting the products to test is a multi-dimensional problem. Indeed, the testing process usually involves multiple and possibly conflicting objectives, like generating the products to test with respect to t-wise but at the same time minimizing the number of products to test, maximizing the number of optional features exercised and minimizing the cost of testing these products. Such requirements necessitate a trade-off between several testing objectives. However, most of the approaches selecting relevant products to be tested target a single objective at a time. While this may be sufficient in certain cases, this method does not reflect real-life testing situations and constraints. In view of this, this paper proposes a genetic algorithm to

handle multiple test objectives for SPLs. The approach is validated through a case study conducted on 8 FMs.

The remainder of this paper is organized as follows. Section 2 presents the concepts and the motivation underlying the proposed approach. Section 3 details the introduced algorithm to solve multi-objective test generation for SPLs. Section 4 reports on experiments. Finally, Section 5 discusses related work and Section 6 concludes the paper.

2. BACKGROUND AND MOTIVATION

This section introduces some notations and concepts used in this paper and motivates the proposed approach.

2.1 Feature Models and Products

Feature Models (FMs) represent the features and the constraints linking the features of a SPL. They are usually represented through a Feature Diagram (FD) [6]. Figure 1 depicts an example of FD of a mobile phone product line of 10 features and the dependencies between the features [2]. For instance, the presence of the *Camera* feature requires the *High resolution* one to be present too.

A FM can easily be translated to logic [23]. Doing so results in a boolean formula where its variables represent the features of the FM. A satisfiability (SAT) solver can then be used to generate valid configurations by using this formula. A configuration of a FM is a set of features that are proposed by the software product. The variables of the formula are assigned *true* for the selected features of the product and *false* for the others. The configuration is said to be valid if the formula is satisfied, i.e. its evaluation is *true*. For instance, the following configuration satisfies the formula of the example FM:

$$C = \{Mobile\ Phone, Calls, Screen, Basic\}.$$

In this configuration, the *Mobile phone*, *Calls* and *Basic screen* variables are assigned to *true*, while the other variables are assigned to *false*. Therefore, the configuration is valid since all the constraints of the FM are fulfilled.

We define as a *product* the set of selected and unselected features that characterize this product [12]. Thus, if we consider a FM of n features, one product P is represented as $\{\pm f_1, ..., \pm f_n\}$, where $+f_i$ indicates a feature which is selected by this product, and $-f_i$ an unselected one. Like for configurations, a product is said to be valid if it satisfies the constraints of the FM, i.e. its formula. For instance, the following product P associated to the configuration C is valid:

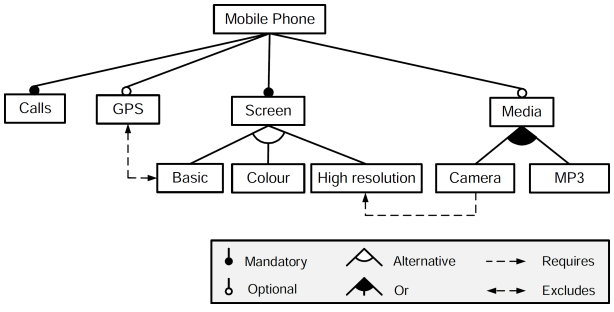


Figure 1: A Feature Diagram of a Mobile Phone Product Line [2].

$$P = \{+Mobile\ Phone, +Calls, -GPS, +Screen, +Basic, -Colour, -High\ Resolution, -Media, -Camera, -MP3\}.$$

2.2 Test Suites, Cost and Pairwise Coverage

In this paper, a product corresponds to a *test case* of a SPL and a *test suite* to a set of m products. To simplify, we will refer to products as *tests*.

In an attempt to take into account the *testing cost* of products, some assumptions are made. We assume that the testing effort of each product is related to the number of features that it contains. Additionally, each feature requires a different amount of resources in order to be tested. To this end, a value representing an estimate of its testing cost is assigned to each feature. Thus, the cost of testing one product is assumed to be equal to the sum of the cost of the features that it is composed of. More formally, if c_i denotes the cost of the feature f_i, a product $P_j = \{\pm f_1, ..., \pm f_n\}$ has a cost C_t equals to:

$$C_t(P_j) = \sum_{i=1}^{n} p(i)c(i), \text{ where } p(i) = \begin{cases} 1 & \text{if } +f_i \\ 0 & \text{if } -f_i \end{cases}.$$

The cost of a test suite is the sum of the cost of the m products that it is composed of. Thus, the cost of $x = \{P_1, ..., P_m\}$ is given by:

$$cost(x) = \sum_{j=1}^{m} C_t(P_j).$$

Putting a cost to each feature transforms the initial FM into an attributed FM [24]. A similar way of representing the cost of features and products can be found in [24].

Finally, pairwise is a testing technique that focuses on the interactions between any two features of a SPL [26]. Such an interaction is called a pair of features. Pairwise coverage denotes the ability of a test suite to cover all the pairs of features that exist in a SPL. In this paper, pairwise coverage is measured as the number of pairs covered by the products of the test suite.

2.3 Motivation

In practice, software development introduces several constraints on the actual testing process [7]. Constraints like occupying a specific amount of resources, meeting a specific budget or finishing the process on time are such examples. To meet these needs, it may be necessary to suspend the testing process before testing all the selected products. In such a case, it is required to minimize the process cost by maximizing the level of testing thoroughness at the same time. This is a difficult task due to the following two problems to face with: ***a)*, in the SPL context, not all the products (as defined in Section 2.1) are necessarily valid**. This fact, makes the test selection a hard problem [25]. The problem is so hard that most of the existing approaches are encountering difficulties in selecting products to test [25]. ***b)*, among the valid products, the aforementioned objectives are competitive, i.e. minimizing the cost reduces the quality of testing and vice versa**. This is analogous to test suite minimization, which is known to be a hard problem [29] and it is clearly escalated in the presence of the *a)* problem.

Addressing both these problems at the same time makes the test selection task challenging. In addition, the practical need of dealing with these problems simultaneously motivates the suggestion of such an approach. In view of this, the

present paper introduces a novel approach avoiding invalid products and capable of dealing with multiple objectives at the same time. Indeed, it makes a combined use of constraint solving and multi-objective optimization techniques to fulfill the test selection goals. The proposed approach is validated on a set of real feature models dealing with the following objectives:

1. Maximizing the pairwise coverage,
2. Minimizing the number of products selected,
3. Minimizing the overall test suite cost (as defined in Section 2.2).

A conducted case study reveals the effectiveness and the practicality of the proposed approach. In particular, it is capable of selecting test suites with higher pairwise coverage than randomly selected ones containing the same number of products. Additionally, it is able to generate test suites composed of less products and having a lower cost than randomly selected suites having the same pairwise coverage.

3. A GENETIC ALGORITHM FOR MULTI-OBJECTIVE TEST GENERATION

Exhaustive SPL testing is difficult to achieve in practice since it involves the test of thousands, even millions of products. As a result, it is necessary to select the products matching the testing objectives. Looking over the space of all the possible products for the optimal solution that satisfies the most these objectives is not feasible due to the large size of the space. Indeed, it is not possible to evaluate all the solutions. Genetic algorithms form a family of techniques that have been proven to be quite effective in finding solutions on large search spaces [11]. The approach introduced in this paper is a genetic algorithm that uses a SAT solver to only search the space of valid products, thus pruning the invalid ones.

3.1 Genetic Algorithms

Genetic algorithms form search-based heuristics mimicking the natural evolution process. They represent a smart way to randomly search for solutions to optimization problems. To apply such an approach, several parameters like the genes, the individuals and the objective function have to be defined. The individuals correspond to what composes a possible solution to the optimization problem. Each individual is composed of several units, called genes and the set of individuals that is handled by the algorithm is called the population. The objective function quantifies the individuals' ability to solve the optimization problem. Generally, these algorithms operate by repeatedly reproducing, adjusting and selecting the best individuals of the population. Based on the process presented in the following subsection, the population is gradually evolved by optimizing the solutions it encodes.

3.1.1 The Process

Genetic algorithms operate by evolving a population. The evolution is guided by an objective function. The initial population is usually produced at random and evolved based on a given set of operations on its individuals. Usually, three operations are used for the evolution of the population. These are the selection, crossover and mutation [11].

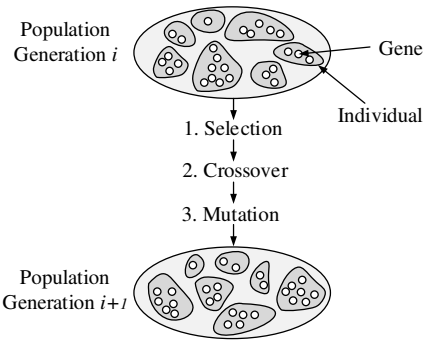


Figure 2: The process of evolving a population.

1. Selection chooses individuals for performing crossover and mutation. The selection is made by choosing the individuals with the best scores according to the objective function.
2. Crossover selects two individuals and switches some of their genes. This is usually performed by ordering the individuals' genes and switching all the genes after a randomly selected point. Crossover results into two new individuals called offsprings.
3. Mutation performs on an offspring by changing the values of one or more of its genes.

Performing the selection, crossover and mutation operations on a population results in one evolution cycle of the population. This cycle is called population generation. An overview of one generation is presented in Figure 2. The algorithm terminates after completing a predefined number of generations.

3.1.2 Search for Multi-Objective Optimization

Searching for optimizing more than one objective at the same time is usually referred to as multi-objective optimization. The aim of these approaches is to search for optimal (or nearly optimal) solutions requiring trade-offs between two or more conflicting objectives. In the present paper, our multi-objective optimization is defined as follows:

Let X_p be the set of all the possible products of a SPL and let $x = \{P_1, ..., P_m\}$ be a set of m products.
Given: a FM, a given amount of time or iterations, t, and the vector of k objective functions $\mathbf{F}(\mathbf{x}) = [F_1(x), ..., F_k(x)]^T$ where each objective function F_i is a normalized function to minimize of the form $F_i(x) : P_1, ..., P_m \rightarrow \mathbb{R}$.
Problem: finding $x \in X_p$ with respect to t such as $\min_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.
The minimization of $\mathbf{F}$ is is the process of optimizing systematically and simultaneously the k objective functions [20].

3.2 Multi-Objective Test Generation for Software Product Lines

The proposed approach is a multi-objective genetic algorithm. Like any genetic algorithm, it requires the definition of its ingredients (genes, individuals and population), its operations (selection, crossover and mutation) and the objective function that evaluates how each individual fits to the problem.

3.2.1 Modeling Individuals and Population

A solution to our problem is a set of products that gives the maximum pairwise coverage with the minimum cost and number of products. To fit the problem with the genetic algorithm, it is needed to model the population, the individuals and the genes in terms of the actual problem. Therefore, since an individual represents a possible solution to the problem, it can be modeled as a set of products $x = \{P_1, ..., P_m\}$. Thus, each valid product represents a gene and the set of individuals handled by the genetic algorithm represents the population. This allows forming as a search space all the possible sets of valid products. This represents a huge space due to the intractable number of the possible products contained in a SPL.

However, enabling a search approach over this space cannot be performed directly. Recall that not all the products of a SPL form valid ones. Therefore, there is a need to efficiently deal with the invalid products. This is not an easy task due to the large number of invalid products, especially for large SPLs [25]. To overcome this difficulty, a SAT solver is used to provide random valid products. This is achieved by randomizing the solutions' enumeration order of the FM's formula [19]. To this end, random products, sets of random products and the initial population can be produced efficiently [12]. Thus, the search space is reduced to only include valid products. The importance of this step is that it prunes the invalid products from the search space.

3.2.2 Modeling Genetic Algorithm Operations

Crossover is an operation defined between two selected individuals, called the parents and it is performed as depicted by Figure 3. This operation is performed by selecting l products from the smallest in size parent and swapping them with randomly selected ones from the other (bigger in size) parent. Our individuals form sets of products and thus the order of the genes does not matter. Hence, swapping randomly some products is equivalent to the usual crossover operation. Additionally, doing so ensures that the individuals are having the same sizes during the whole evolution process. Crossover operation results in two offsprings. These are then mutated according to the mutation operation as depicted by Figure 4. In *mutation* operation, a product is randomly selected from the individual and replaced by a randomly selected product (from the space of all the valid products).

Besides the above operations, the proposed approach incorporates two additional operations. These are the elitism and diversify operations. *Elitism* selects the best e individuals of one population and includes them directly to the new one. *Diversify* operation adds one new individual, randomly produced directly into the new population. This ensures the diversity of the population individuals during the evolution process.

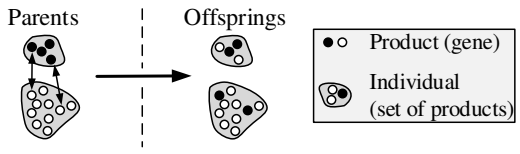


Figure 3: Crossover operation. A random number of l products are selected in the smallest parent. Each of them is swapped with a random product selected from the other parent to produce the two offsprings.

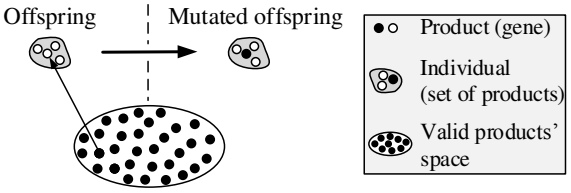


Figure 4: Mutation operation. A random product is selected from the offspring and replaced by a product randomly selected from the space of all the products valid towards the Feature Model.

3.2.3 The Objective Function

The proposed approach is based on an objective function $\mathbf{F}(x)$, specially designed for the SPL testing context. As introduced in Section 3.1.2, $\mathbf{F}$ is a vector composed of the following $k = 3$ objective functions F_1, F_2 and F_3.

1. Maximization of the pairwise coverage. This objective aims at ensuring that the selected products have the highest possible level of pairwise coverage:

$$F_1(x) = cov(x),$$

   where *cov* is a function that evaluates the number of pair of features covered by $x = \{P_1, ..., P_m\}$.

2. Minimization of the number of products. Here, the objective is to test the minimum number of products:

$$F_2(x) = card(x),$$

   where *card* is a function that returns the number of products m of $x = \{P_1, ..., P_m\}$.

3. Minimization of the testing cost. This objective function aims at minimizing the cost of testing the products:

$$F_3(x) = cost(x),$$

   where *cost* is a function returning the cost of testing these $x = \{P_1, ..., P_m\}$ products.

In order to evaluate $\mathbf{F}$, each objective function is normalized so that they have the same magnitude using the following formula [20]:

$$\frac{F_i(x) - F_i^*}{F_i^{\max} - F_i^*},$$

where F_i^* is the utopia point and $F_i^{\max}$ the maximum objective functions values. In addition, the objective max F_1 is transformed into a minimization problem $\min(-F_1)$ in order to deal with minimization problems only. As a result, each objective function returns a value that holds between 0 and 1, where 0 means that the objective is perfectly fulfilled. To evaluate $\mathbf{F}$, each function is assigned a weight w_j, where $\sum_{j=1}^{k} wj = 1$. Thus, the fitness of each individual $I = \{P_1, ..., P_m\}$ is computed using a weighted sum as follows:

$$\mathbf{F}(I) = \textstyle\sum_{j=1}^{k} w_j F_j(I).$$

3.3 Algorithm

The technique is formalized in Algorithm 1. Informally, this approach starts by creating an initial random population (lines 3 to 14). The size of the population is specified by the user as long as the maximum size of an individual. Each individual is a set of $1, ..., m$ products randomly selected from the space of all the products that are valid towards the FM (lines 6 to 11). The objective function is then evaluated for each individual of the initial population (line 12).

The second step of the algorithm is the evolution of the population into a new one (lines 15 to 46). First, the elitism operation is performed (lines 17 to 19). Then, one random individual is created, evaluated and added to the new population to ensure having new products (lines 20 to 27). This is the diversity operation. The next steps ares crossover and mutation. To complete the new population until reaching its size n, individuals are created using crossover and mutation operators. The crossover (lines 29 and 30) aims at creating two offsprings from their selected parents. The two parents are selected using a fitness proportionate selection, also known as roulette wheel selection. The mutation occurs on the offsprings with a certain probability fixed by the user (lines 32 to 37). The fitness of these two offsprings is then evaluated and these two new individuals are added to the new population (lines 39 to 40).

Finally, the new population is reduced to the initial population size (lines 42 to 44) and the current population is replaced by the new one (line 45) and it continues to the next generation (line 46). When the algorithm terminates, the individual that has the best fitness, i.e. the one with the minimum $\mathbf{F}$ value is returned (line 47).

Algorithm 1 Multi-objective Test Generation

1: **input:** $t, n, m, e < n, \mathbb{P}_{\text{mutation}}, w_1, .., w_k, fm$ ▷ t is the time or number of iterations, n is the population size, m is the maximum individual size, e is the number of individuals involved in elitism, $\mathbb{P}_{\text{mutation}}$ is the probability to mutate one individual, $w_1, .., w_k$ are the respective weights of each objective function $F_1, .., F_k$ and fm is the FM.
2: **output:** $x = \{P_1, ..., P_m\}$ ▷ Solution (an Individual)
3: $x \leftarrow \emptyset$
4: $pop \leftarrow \emptyset$ ▷ Population is a set of individuals
5: **while** $card(pop) < n$ **do**
6: $s \leftarrow$ random integer from $1, ..., m$
7: $I \leftarrow \emptyset$ ▷ An individual is a set of s products
8: **while** $card(I) < s$ **do**
9: $P \leftarrow$ random product(fm) ▷ Using a SAT solver
10: $I \leftarrow I \cup \{P\}$
11: **end while**
12: Evaluate $\mathbf{F}(I) = \sum_{j=1}^{k} w_j F_j(I)$
13: $pop \leftarrow pop \cup \{I\}$
14: **end while**
15: **while** elapsed time or number of iterations $< t$ **do**
16: $newPop \leftarrow \emptyset$
17: **while** $card(newPop) < e$ **do**
18: $newPop \leftarrow newPop \cup \{\{I\} \mid I \in pop \wedge I \notin newPop \wedge \min \mathbf{F}(I)\}$
19: **end while**
20: $s \leftarrow$ random integer from $1, ..., m$
21: $I \leftarrow \emptyset$
22: **while** $card(I) < s$ **do**
23: $P \leftarrow$ random product(fm) ▷ Using a SAT solver
24: $I \leftarrow I \cup \{P\}$
25: **end while**
26: Evaluate $\mathbf{F}(I) = \sum_{j=1}^{k} w_j F_j(I)$
27: $newPop \leftarrow newPop \cup \{I\}$
28: **while** $card(newPop) < n$ **do**
29: $I_{\text{parent1}}, I_{\text{parent2}} \leftarrow selection(pop)$ ▷ Selected according to a fitness proportionate selection method
30: $I_{\text{child1}}, I_{\text{child2}} \leftarrow crossover(I_{\text{parent1}}, I_{\text{parent2}})$
31: $p_1, p_2 \leftarrow$ random real number from $[1, 2]$
32: **if** $p_1 \leq \mathbb{P}_{\text{mutation}}$ **then**
33: $mutate(I_{\text{child1}})$
34: **end if**
35: **if** $p_2 \leq \mathbb{P}_{\text{mutation}}$ **then**
36: $mutate(I_{\text{child2}})$
37: **end if**
38: Evaluate $\mathbf{F}(I_{\text{child1}}) = \sum_{j=1}^{k} w_j F_j(I_{\text{child1}})$
39: Evaluate $\mathbf{F}(I_{\text{child2}}) = \sum_{j=1}^{k} w_j F_j(I_{\text{child2}})$
40: $newPop \leftarrow newPop \cup \{I_{\text{child1}})\} \cup \{I_{\text{child2}})\}$
41: **end while**
42: **while** $card(newPop) > n$ **do**
43: $newPop \leftarrow newPop \setminus \{\{I\} \mid I \in newPop \wedge \max \mathbf{F}(I)\}$
44: **end while**
45: $pop \leftarrow newPop$
46: **end while**
47: $x \leftarrow I \mid I \in pop \wedge \min \mathbf{F}(\mathbf{I})$
48: **return** x

4. CASE STUDY

In this section, the proposed multi-objective test generation approach is assessed on a set of FMs. The objective of this case study is to answer the two following research questions:

- [RQ1] *Is* $\mathbf{F}$ *capable of leading to a fulfillment of the three objectives? In other words, does the minimization of* $\mathbf{F}$ *results in a maximization of* F_1 *(or a mininimization of* $(-F_1)$*), a minimization of* F_2 *and a minimization of* F_3*?*
- [RQ2] *How does the multi-objective generation technique compares with a random one?*

Answering the first question amounts to evaluate whether the objective function $\mathbf{F}$ is capable of improving the studied objectives. We expect to see a decreasing trend in all three objectives in relation to population generations. In practice, this means that a better trade-off can be achieved. This trade-off leads to a higher pairwise coverage, less products and a lower cost. Since no other approach takes into account these objectives at the same time, our second question aims at comparing the two of them when keeping the other one set, to enable the comparison with random test suite generation. Hence, we select random product sets of a) the same size and b) achieving the same pairwise coverage as our approach. If, for the same number of products, our approach achieves to provide a lower cost and higher pairwise coverage than the random set, we can consider it as being a better one. Similarly, it will be successful if it provides less products and a lower cost than random for a certain level of pairwise coverage.

To answer these questions, an experiment composed of 8 FMs of varying sizes was conducted. We applied our approach on these FMs to evaluate the population evolution and to compare it with a random approach. All the employed FMs were taken from the Software Product Line Online Tools (SPLOT) repository [22] and have been widely used in literature. The FMs details are recorded in Table 1. For each subject FM, the number of features, the number of products that can be configured and the number of valid pairs are presented. For each FM, we randomly assigned a value between 1 and 10 to all non-mandatory features to represent the cost value of the features, as presented in Section 2.2. Further details on the conducted experiment are given in the following subsections.

Table 1: Feature Models used in the case study. It presents, for each of them, the number of features, the number of possible valid products that can be configured and the total number of valid pairs.

	Counter Strike Simple FM	DS Sample	SPL SimulES, PnP	Electronic Drum	Smart Home v2.2	Video Player	Model Transformation	Coche Ecologico
Number of features	24	32	41	52	60	71	88	94
Number of products (≈)	18,176	73,728	6,912	331,776	3.87×10^{9}	4.5×10^{13}	1.65×10^{13}	2.32×10^{7}
Number of pairs	833	1,448	2,592	3,746	6,189	7,528	13,139	11,075

4.1 Algorithm Parameters

Since objective F_1 results in selecting a higher number of products and F_2, F_3 results in selecting a lower number of products, we assigned the following weights: $w_1 = 0.5$ for F_1 and $w_2 = w_3 = 0.25$ for F_2 and F_3. This assignment represents the balanced between the studied objectives as set for our experiment. It is noted that our approach is not limited to this balance. Thus, the tester may set a different balance according to his needs. The population size has been set to $n = 100$ and the maximum size of an individual (a potential solution) has been set to $m = 100$. The mutation probability $\mathbb{P}_{\text{mutation}}$ has been set up to 0.05 and the elitism value e to 5. Finally, the approach has been limited to run for $t = 500$ generations.

4.2 Evaluation of Objective Function F (RQ1)

4.2.1 Setup

We performed the multi-objective test generation 30 times per FM using the above-mentioned parameters. For each of the 30 runs, we measured the initial values (at generation 1) and the final values (at generation 500) of both the 3 sub-objectives and the objective **F**.

To evaluate whether these differences are statistically significant, we followed the guidelines suggested by Arcuri and Briand in [1] by performing a Mann-Whitney U Test. It is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other. We obtain from this test a probability called p-value which represents the probability that the two samples are equal. It is conventional in statistics to consider that the difference is not significant if the p-value is higher than the 5% level. The experiments involving this statistical test used two-tailed tests.

4.2.2 Results

Table 2 presents per FM the average values on the 30 runs for each of the objective and for **F**. F_1 is the number of pairs covered by the generated products (to maximize), F_2 is the number of products (to minimize) and F_3 is the cost of testing the generated products (to minimize). **F** is the compromised between the 3 objectives. From this table, one may observe that final values of both the 3 objectives and the objective function are better than initial one, i.e. decreasing for F_2, F_3 and **F** and increasing for F_1 since it's a maximization. This underlines that a decreasing in **F** leads to a better fulfillment of each objective. These difference are most of the time statistically significant with p-values lower than 0.05 or highly significant with p-values lower than 0.001, fact which demonstrates the appropriateness of the objective function with only 500 generations of the algorithm.

Besides, Figure 5 depicts the evolution of the objective function **F** and the normalized objective function over the generations of the algorithm. Since all the three objectives are transformed into minimization problems, i.e. lower values of the objective functions represent better solutions to the problem, this figure clearly shows the decreasing trend of each objective function. It therefore demonstrates that **F** leads to a better solution regarding all the examined objectives.

4.2.3 Answering RQ1

The results presented in the previous section clearly show the ability of the objective function to fulfill the three studied objectives. In particular, **F** is capable of finding better solutions for all the objectives under investigation. While some differences may not be statistically significant, recall that the approach is a compromise between the conflicting objectives. It therefore tends to compromise the 3 objectives according the w_1, w_2 and w_3 parameters. The overall

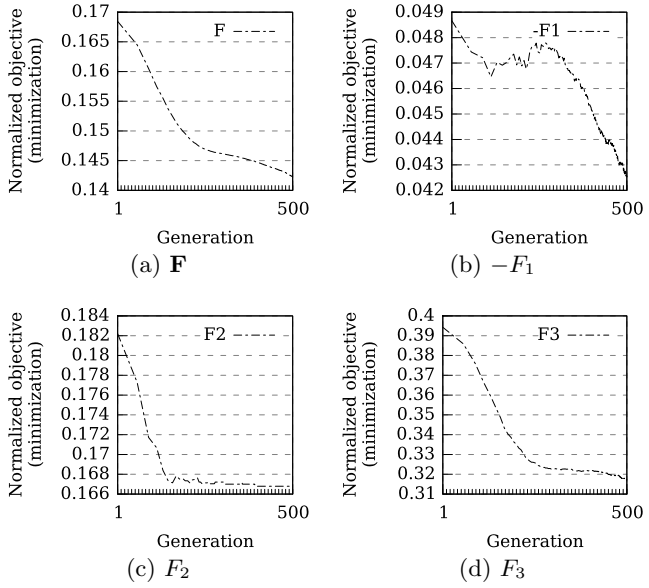


Figure 5: Evolution of the objective function F and each normalized sub-objectives (to be minimized) during the 500 generations of the multi-objective test generation approach.

Table 2: Evolution of the objective function F and the sub-objectives from the initial generation of the multi-objective approach to the final one (500 generations). The final and initial values are the average between the 30 runs. The p-value is the results of the Mann-Whitney U Test between the 30 first initial values and the 30 final ones.

	F			F_1: pairwise coverage (to maximize)			F_2: # products (to minimize)			F_3: cost (to minimize)		
	Initial	Final	p-value	Initial	Final	p-value	Initial	Final	p-value	Initial	Final	p-value
Counter Strike Simple FM	0.163	0.115	<0.001	819.7	819.63	0.79	15.46	14.466	0.176	658.13	369.90	<0.001
SPL SimulES, PnP	0.163	0.136	<0.001	1431.4	1439.03	0.003	14.33	12.8	<0.001	906.73	680.66	<0.001
DS Sample	0.189	0.172	<0.001	2364.2	2382.9	0.07	31.866	27.7	<0.001	1040.2	887.96	<0.001
Electronic Drum	0.146	0.132	<0.001	3633.6	3665.06	<0.001	18.7	17.4	0.04	1221.96	1079.6	0.001
Smart Home v2.2	0.177	0.138	<0.001	6041.46	6056.66	0.60	17.7	17.03	0.33	2282.86	1537.46	<0.001
Video Player	0.162	0.135	<0.001	7430.66	7428.76	0.20	15.13	13.86	0.011	2000.63	1443.86	<0.001
Model Transformation	0.175	0.153	<0.001	12733.73	12788.1	0.387	17.96	17.16	0.48	3522.5	2829.36	<0.001
Coche Ecologic	0.169	0.154	<0.001	10560.26	10618.06	0.039	21.13	19.66	0.19	2083.1	1761.63	<0.001

objective **F** has always highly statistically significance difference, showing that **F** clearly guides the population generation. Finally, it must be mentioned that the proposed approach achieves the above results using only a small number of generations (500 generations). This can be viewed as an achievement of the approach since search-based approaches do require thousands of executions in order to be effective [11].

4.3 Comparison with Random (RQ2)

4.3.1 Setup

To assess our approach, we compared it with a baseline. To do so, we used two baseline comparison basis. In the first one, we selected random sets of products having the same F_1 value as our approach. In the second one we selected random product sets having the same F_2 value as our approach. The F_1 comparison basis aims at evaluating how many products for which cost are provided by the examined approaches (baseline and proposed) to achieve the same level of pairwise coverage. The F_2 comparison basis evaluates the pairwise coverage and the cost induced by the generated products for the same number of products. In the end, for each run of our approach, two random runs have been performed: the first one by setting F_1 as the comparison basis and the second one using F_2. The conducted experiment (including both the baseline and the proposed approach) was independently repeated 30 times.

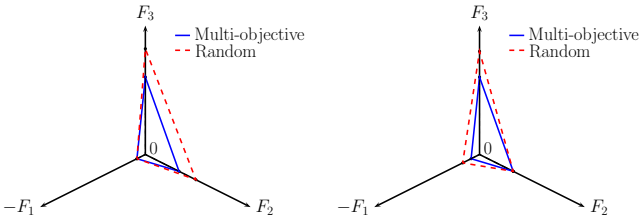
To evaluate whether the differences are significant, we performed a Mann-Whitney U Test, as presented in Section 4.2.1. For each comparison (**F**, F_2 and F_3 on F_1 comparison basis and **F**, F_1 and F_3 on F_2 comparison basis), we got one p-value per FM, i.e. 8 in the total. Each p-value results from the comparison between the 30 values obtained on the 30 runs by the proposed approach with those obtained at random.

4.3.2 Results

Table 3 records the comparison between our test generation approach and the baseline one based on the F_1 and the F_2 comparison basis. For each FM and each comparison basis, the average, minimum and maximum values of **F** and the three objectives F_1, F_2 and F_3 one the 30 runs are presented. F_1 represents the number of pairs covered by the generated products (to be maximized), F_2 represents the number of products (to be minimized) and F_3 represents the cost of testing the generated products (to be minimized). From this table, it is clear that the proposed approach performs better than a random one. For instance, for the Smart Home v2.2 FM on the F_1 comparison basis (i.e. for achieving the same pairwise coverage), the proposed approach proposes on average around 17 products with a cost of $\approx$ 1,537 where a random technique requires around 22 products with a cost of $\approx$ 3,016.

Figure 6 depicts the achieved values for each objective for both basis of comparison. The values are the average on all the FMs for the 30 runs. Here, the smallest triangle signifies a better solution since each normalized objective is a function to be minimized. An objective value equals to 0 means that this objective is perfectly fulfilled. The length of the axis of each objective is 1. This figure shows that a) for the same pairwise coverage, the proposed approach requires less products with a lower cost and b) for the same number of products, our approach provides a higher pairwise coverage and a lower cost compared to random products.

Finally, the results of the statistical test are depicted by Figure 7. It presents, for each comparison, the distribution of the 8 p-values (one per FM). Each p-value is the result of the comparison between the 30 values obtained for each objective during each run of the proposed and random approaches. From this figure, one may observe that all the p-values are lower to 0.001, fact which denotes the a high statistical difference between the results achieved by our approach compared to the results of the random one.



(a) Same pairwise coverage (F_1 basis) (b) Same number of products (F_2 basis)

Figure 6: Normalized sub-objectives (to be minimized) according to F_1 and F_2 comparison basis. Values closer to 0 represent better solutions. These values are the average on all the FMs for all the 30 runs of each approach. The length of each axis is 1.

Table 3: Comparison between the multi-objective test generation approach and the random one. For each Feature Model, the values of the objective functions studied and F are represented. F_1 is the number of pairs (to be maximized), F_2 is the number of products (to be minimized) and F_3 is the cost (to be minimized). The comparison with random has been made by fixing either F_1 or F_2 on 30 runs per approach.

			Multi-objective test generation			Random test generation		
			avg	min	max	avg	min	max
Counter Strike Simple FM	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.115	0.107	0.122	0.173	0.154	0.2
		F_2 (to min.)	14.466	11	17	18.833	12	28
		F_3 (to min.)	369.9	276	440	832.433	512	1,218
	Same number of products (F_2 basis)	**F** (to min.)	0.115	0.107	0.122	0.181	0.155	0.238
		F_1 (to max.)	819.633	803	828	806.833	763	828
		F_3 (to min.)	369.9	276	440	665.13	510	889
SPL SimuelES, PnP	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.136	0.13	0.14	0.173	0.159	0.187
		F_2 (to min.)	12.8	10	15	18.966	14	26
		F_3 (to min.)	680.666	567	813	1,234.8	940	1,656
	Same number of products (F_2 basis)	**F** (to min.)	0.136	0.13	0.14	0.177	0.16	0.20
		F_1 (to max.)	1,439.033	1,429	1,446	1,411.066	1,367	1,445
		F_3 (to min.)	680.666	567	813	859.333	680	1,039
DS Sample	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.172	0.169	0.177	0.214	0.182	0.302
		F_2 (to min.)	27.7	22	34	44.966	32	83
		F_3 (to min.)	887.966	700	1,106	1,469.8	1,024	2,716
	Same number of products (F_2 basis)	**F** (to min.)	0.172	0.169	0.177	0.214	0.195	0.246
		F_1 (to max.)	2,382.9	2,328	2,428	2,236.366	2,093	2,362
		F_3 (to min.)	887.966	700	1,106	902.9	725	1,121
Electronic Drum	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.132	0.130	0.133	0.155	0.142	0.173
		F_2 (to min.)	17,4	14	20	24.5	17	32
		F_3 (to min.)	1,079.6	872	1,255	1,645.533	1.165	2,210
	Same number of products (F_2 basis)	**F** (to min.)	0.132	0.130	0.133	0.155	0.144	0.180
		F_1 (to max.)	3,665	3,628	3,693	3,585.133	3,458	3,661
		F_3 (to min.)	1,079.6	872	1,255	1,174.7	926	1,367
Smart Home v2.2	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.138	0.133	0.144	0.191	0.166	0.234
		F_2 (to min.)	17.033	12	20	21.966	15	36
		F_3 (to min.)	1,537.466	1,195	1,836	3,016.533	1,974	5,184
	Same number of products (F_2 basis)	**F** (to min.)	0.138	0.133	0.144	0.19	0.166	0.223
		F_1 (to max.)	6,056.666	5,973	6,107	5,976	5,756	6,087
		F_3 (to min.)	1,537.466	1,195	1,836	2,330.4	1,532	2,872
Video Player	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.135	0.128	0.138	0.167	0.152	0.188
		F_2 (to min.)	13.866	11	16	16.5	14	24
		F_3 (to min.)	1,443.866	1,230	1,687	2,2236.5	1,858	3,339
	Same number of products (F_2 basis)	**F** (to min.)	0.135	0.128	0.138	0.173	0.159	0.208
		F_1 (to max.)	7,428.766	7,739	7,468	7,341.233	6,925	7.471
		F_3 (to min.)	1,443.866	1,230	1,687	1907.433	1,467	2,444
Model Transformation	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.153	0.149	0.158	0.185	0.176	0.199
		F_2 (to min.)	17.166	14	21	20,733	18	25
		F_3 (to min.)	2,829.366	2,353	3,319	4,325.166	3,588	5,304
	Same number of products (F_2 basis)	**F** (to min.)	0.153	0.149	0.158	0.187	0.174	0.199
		F_1 (to max.)	12,788.1	12,657	12,902	12,595.3	12,262	12,815
		F_3 (to min.)	2,829.366	2,353	3,319	3,556.166	2,742	4,509
Coche Ecologico	Same pairwise coverage (F_1 basis)	**F** (to min.)	0.154	0.151	1.158	0.186	0.172	0.217
		F_2 (to min.)	19.666	16	24	29.3	20	41
		F_3 (to min.)	1,761.633	1383	2153	2,984,3	2,051	4,384
	Same number of products (F_2 basis)	**F** (to min.)	0.154	0.151	1.158	0.188	0.167	0.207
		F_1 (to max.)	10,618	10,492	10,726	10,302	10,040	10,553
		F_3 (to min.)	1,761.633	1383	2153	2,016.333	1,631	2,404

4.3.3 Answering RQ2

We compared the multi-objective test generation approach with a baseline technique using F_1 and F_2 as a basis comparison. In all the cases, the objectives are better fulfilled by our approach, as demonstrated by the results of Section 4.3.2. Overall, for the same pairwise coverage, the approach selects less products with a lower cost. For the same number of products, the approach provides a higher pairwise coverage at a lower cost. In addition, the differences between the objectives values reach by our technique and the values reached by the baseline are statistically highly significant, fact which demonstrates the effectiveness of the approach.

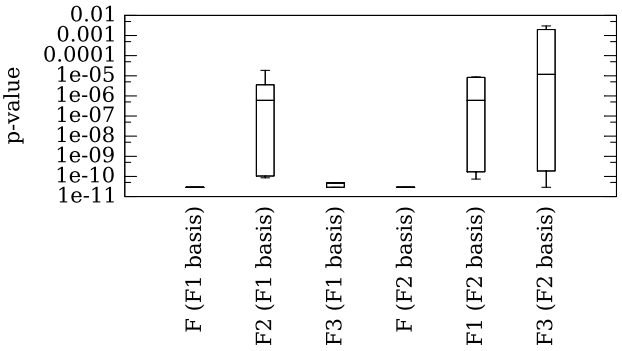


Figure 7: Distribution of the p-values for the comparison with random. For each comparison ($\mathbf{F}, F_2$ and F_3 on F_1 comparison basis and $\mathbf{F}, F_1$ and F_3 on F_2 comparison basis), the 8 p-values (one per Feature Model) are represented with a boxplot. Each p-value has been obtained by comparing the 30 values obtained on the 30 runs for each approach.

4.3.4 Threats to Validity

The conducted experiments involve potential threats to validity. First, there is a threat regarding the generalization of the results reported in this study. Indeed, a different set of FMs might output different results. We used a set of 8 FMs widely used in literature with different size and level of complexity to reduce this threat and to ensure that the FMs used form a good sample.

Additional threats can be identified due to a) to our implementation, which might contains errors that can affect the presented results and b) the performed experiments. To overcome this issue, we divided our implementation into subroutines to minimize the potential errors and we make it publicly available. We also repeated the conducted experiments independently for 30 times to avoid any risk due to random effects, like the fortunate selection of the (nearest) optimal solution.

5. RELATED WORK

Test generation is an issue than has been investigated by the research community since the last decades [8].

With respect to constraint solving, Johansen *et al.* [15] proposed a covering array technique to generate the products covering all the t-wise interactions between features. In the same lines, a search-based approach that achieves scalable but partial t-wise coverage has been introduced in [12]. Perrouin *et al.* [26] proposed a method based on the Alloy SAT solver to generate t-wise test suites. Modeling the importance of t-wise interactions have been proposed in [16] by putting weights on interactions and measuring weight coverage. Like the approach presented in this paper, these methods have been applied in conjunction with a SAT solver but aim at fulfilling only one objective at a time, i.e. either the t-wise coverage or the weight coverage. Other work, e.g. [13, 14] used a SAT solver to only generate valid products. Here, maximizing the 2-wise coverage of the generated products is one of the 3 objectives to fulfill simultaneously. Moreover, we combine constraint solving with search-based techniques.

Regarding evolutionary algorithms, Konak *et al.* [18] proposed a tutorial on the use of these kind of methods for multi-objective optimization purposes. Ensan *et al.* [9] proposed a genetic algorithm approach where each gene is a feature. The crossover can thus produce invalid products. Furthermore, the explored space may contain invalid products. Their fitness function measures coverage by evaluating the variability points to be bound and the constraints concerned by the features of a product. In our approach, we use a SAT solver to only explore the space containing products valid towards the FM. The modeling of genes is performed at the product level and the crossover and mutation operators introduced avoid the introduction of invalid products.

Finally, in the context of multi-objective optimization, Olaechea *et al.* [24] introduced a tool that supports multi-objective goals in the configuration of features. The approach works on attributed feature models which provide quality attributes to features. Their technique use an exact solving of the multi-optimization problem and consider FMs with one to 3 objectives, where the one with 3 objectives contains 12 features. In our approach, we use an heuristic solving with 3 objectives to fulfill for all the FMs. The advantage is that it allows to scale to large FMs.

6. CONCLUSIONS AND FUTURE WORK

Optimizing different objectives is a hard problem due to the presence of conflicts between them. For example, minimizing the number of tests is in conflict with the maximization of their pairwise coverage since generally more tests lead to a higher coverage. To tackle this problem, the present paper introduces a multi-objective genetic algorithm specially adapted for SPLs. Our approach combines genetic algorithms and constraint solving techniques in a complementary way. Thus, it provides sets of products to test that simultaneously optimize pairwise coverage and testing costs. The work presented in this paper deals with the following issues:

- **Model the test generation problem for SPLs as a search problem**. The proposed approach models products as genes and sets of products as individuals. It also suggests some possible operations on the individuals and an objective function. Therefore, it enables search-based approaches to solve the test generation problem.
- **Use constraint solving technique to prune the invalid products from the search space**. This is a crucial step towards enabling an efficient search process. Our initial experiments show that it is almost impossible to construct sets of valid products without using a constraint solver. Even if this is possible, since invalid products must be removed from the final product sets, they only add barriers to the search process. Thus, it is clear that removing the invalid product greatly reduces the search space and busts the effectiveness of the approach.
- **Propose a genetic algorithm to solve the multi-objective optimization problem**. We propose a generic algorithm for handling the test generation problem according multiple objectives for the context of SPLs. The conducted study show that the approach is practically effective and feasible.

Reproducible tests has been identified as a central tenet of testing [10]. Therefore, to enable the reproducibility of our results, we make the source code of our approach and the data used for the experiments publicly available at:

`http://research.henard.net/SPL/SPLC_2013/`.

Finally, future work includes the following points:

- Conducting additional experiments to further validate the finding of the present paper.
- Investigating the effects of the parameters on the effectiveness and efficiency of the proposed approach.
- Investigating criteria for deciding when to stop the evolution process.
- Applying our approach on real and large scale subjects in order to better quantify the benefits of the approach.

Acknowledgment

The present work is supported by the Fonds National de la Recherche (FNR), Luxembourg, under the projects MITER C10/IS/783852 and MODEL C12/IS/3977071.

7. REFERENCES

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

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, Sept. 2010.

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

[4] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG System: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[5] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *IEEE Softw.*, 15(6):37–45, Nov. 1998.

[6] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. 2000.

[7] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In *FSE*, pages 71–82, 2008.

[8] I. do Carmo Machado, J. D. McGregor, and E. Santana de Almeida. Strategies for testing products in software product lines. *SIGSOFT Softw. Eng. Notes*, 37(6):1–8, Nov. 2012.

[9] F. Ensan, E. Bagheri, and D. Gašević. Evolutionary search-based test generation for software product line feature models. In *CAiSE*, pages 613–628, 2012.

[10] M. D. Ernst. Reproducible tests? non-duplicable results in testing and verification. In *ICST*, April 18–20, 2012.

[11] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Trans. Softw. Eng.*, 36(2):226–247, Mar. 2010.

[12] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines. *CoRR*, abs/1211.5451, 2012.

[13] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. In *ICSTW, A-MOST*, 2013.

[14] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. Towards automated testing and fixing of re-engineered feature models. In *ICSE*, pages 1245–1248, 2013.

[15] M. F. Johansen, O. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *SPLC*, pages 46–55, 2012.

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

[17] K. Kang, J. Lee, and P. Donohoe. Feature-oriented product line engineering. *Software, IEEE*, 19(4):58 – 65, jul/aug 2002.

[18] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, Sept. 2006.

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

[20] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, Apr. 2004.

[21] J. McGregor. Testing a software product line. In *Testing Techniques in Software Engineering*, volume 6153, pages 104–140. 2010.

[22] M. Mendonca, M. Branco, and D. Cowan. S.p.l.o.t.: software product lines online tools, 2009.

[23] M. Mendonca, A. Wasowski, and K. Czarnecki. Sat-based analysis of feature models is easy. In *SPLC*, pages 231–240, 2009.

[24] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside. Modelling and multi-objective optimization of quality attributes in variability-rich software. In *NFPinDSML*, pages 2:1–2:6, 2012.

[25] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. L. Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 20(3-4):605–643, 2012.

[26] 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, 2010.

[27] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. 2005.

[28] R. Rabiser, P. Grunbacher, and D. Dhungana. Supporting product derivation by adapting and augmenting variability models. In *SPLC*, pages 141–150, 2007.

[29] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.