2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Comparative Analysis of Classical Multi-Objective Evolutionary Algorithms and Seeding Strategies for Pairwise Testing of Software Product Lines

Roberto E. Lopez-Herrejon*, Javier Ferrer†, Francisco Chicano†,
Alexander Egyed* and Enrique Alba†
* Software Systems Engineering
Johannes Kepler University Linz, Austria
Email: {roberto.lopez, alexander.egyed}@jku.at
†Universidad de Málaga, Andalucía Tech, Spain
Email:{chicano, ferrer, eat}@lcc.uma.es

***Abstract*—Software Product Lines (SPLs) are families of related software products, each with its own set of feature combinations. Their commonly large number of products poses a unique set of challenges for software testing as it might not be technologically or economically feasible to test of all them individually. SPL pairwise testing aims at selecting a set of products to test such that all possible combinations of two features are covered by at least one selected product. Most approaches for SPL pairwise testing have focused on achieving full coverage of all pairwise feature combinations with the minimum number of products to test. Though useful in many contexts, this single-objective perspective does not reflect the prevailing scenario where software engineers do face trade-offs between the objectives of maximizing the coverage or minimizing the number of products to test. In contrast and to address this need, our work is the first to propose a classical multi-objective formalisation where both objectives are equally important. In this paper, we study the application to SPL pairwise testing of four classical multi-objective evolutionary algorithms. We developed three seeding strategies – techniques that leverage problem domain knowledge – and measured their performance impact on a large and diverse corpus of case studies using two well-known multi-objective quality measures. Our study identifies the performance differences among the algorithms and corroborates that the more domain knowledge leveraged the better the search results. Our findings enable software engineers to select not just one solution (as in the case of single-objective techniques) but instead to select from an array of test suite possibilities the one that best matches the economical and technological constraints of their testing context.**

I. Introduction

Software Product Lines (SPLs) are families of related software products, where each product provides a unique combination of *features* – increments in program functionality [1]. Some of the proven benefits of SPLs are increased software reuse, faster product customization, and reduced time to market [2]. *Feature Models (FMs)* are the *de facto* standard to represent all the valid feature combinations of an SPL [3]. The typically large number of feature combinations in a SPL poses a unique set of challenges for software testing because testing each individual product may not be technically or economically feasible.

Recent surveys and mapping studies on SPL testing [4], [5], attest the increasing relevance of the topic within the research and industrial SPL communities. Salient among the existing approaches are those based on *Combinatorial Interaction Testing (CIT)*, whose premise is to select a group of products where faults due to feature interactions are more likely to occur [6]. In the SPL context, most of the focus has been on *pairwise* testing, that is, on the four possible combinations between any two features[1]. The pairwise feature combinations of a product determine its *coverage*. Thus, pairwise SPL testing aims to select a set of products, referred to as *test suite*, such that their combined coverage contains *all* the possible pairwise feature combinations of the SPL.

Many SPL pairwise testing approaches have been proposed (e.g. [7], [8], [9], [10], [11], [12], [13], [14], [15]). However, these approaches regard SPL pairwise testing as an optimization problem where either coverage (maximize) or test suite size (minimize) are considered the main optimization objective. While this single-objective perspective might be enough for certain limited contexts, it does not capture the more prevailing scenario where, for instance, it might not be feasible or desirable to test all the products of a test suite, or for example when both coverage and test suite size are of equal importance. Thus, the multi-objective perspective enables software developers to analyse the trade-offs between their objectives (sometimes conflicting) and select the test suites that best match their technical and economical constraints. Unfortunately, those pieces of work that have explored a multi-objective perspective in the realm of SPL testing (i.e. [16] and [17]) do use a scalarization technique that flattens all objectives into a single objective by assigning them weights; an approach with proven limitations [18].

In this paper we study the application to SPL pairwise testing of four classical multi-objective evolutionary algorithms: NSGA-II [19], MOCell [20], SPEA2 [21], and PAES [22]. These four algorithms have been extensively and successfully applied to a large number of problem domains and include a diverse set of techniques and concepts of multi-objective evolutionary algorithms. Thus, our first research question is stated as follows: ***RQ1. What is the best algorithm among***

[1]For `A` and `B` features: both selected, both not selected, `A` selected and `B` not, `A` not selected and `B` selected.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

these four for multi-objective SPL pairwise testing?

Furthermore, we analyze the impact of *seeding*, defined by Fraser and Arcuri as any technique that exploits previous related knowledge to help the testing problem at hand [23], on the performance of these algorithms. We developed three different seeding strategies that respectively exploit the knowledge of: *i)* the size of test suites, *ii)* greedily-generated tests suites, and *iii)* the actual test suites obtained using an existing single-objective pairwise testing approach. Hence, our second research question is stated as follows: ***RQ2. How does seeding impact the quality of the solutions obtained by the four algorithms?***

For the evaluation we selected 19 feature models, from diverse problem domains, from different provenance sources, and with different structural characteristics. We compared the performance of the four algorithms on the selected models with the three seeding strategies using two well-known quality indicator measures commonly employed in the multi-objective community, Hypervolume [24] and Generational Distance [25]. In short, our statistical analysis reveals that the third seeding strategy, the one that exploits more domain knowledge, performs better for both of the comparison measures, and identifies the performance differences among the algorithms.

In summary, in the context of SPL pairwise testing our work is the first to: *i)* formalise the SPL pairwise testing problem for classical multi-objective approaches, *ii)* compare classical multi-objective evolutionary algorithms, *iii)* employ well-known multi-objective quality indicators, *iv)* study the impact of seeding, and *v)* evaluate a large and diverse corpus of feature models.

II. Background

In this section, we concisely layout the necessary background where the contributions of this paper take place.

A. Feature Models and Running Example

Feature models have become the *de facto* standard for modelling the common and variable features of an SPL and their relationships collectively forming a tree-like structure. The nodes of the tree are the features which are depicted as labelled boxes, and the edges represent the relationships among them. Feature models denote the set of feature combinations that the products of an SPL can have [3].

Figure 1 shows the feature model of our running example, the *Graph Product Line (GPL)* [26], a standard SPL of basic graph algorithms that has been widely used as a case study in the product line community. In GPL, a product is a collection of algorithms applied to directed or undirected graphs.

In a feature model, each feature (except the root) has one parent feature and can have a set of child features. A child feature can only be included in a feature combination of a valid product if its parent is included as well. The root feature is always included. There are four kinds of feature relationships: i) *Mandatory features* are selected whenever their respective parent feature is selected. They are depicted with a filled circle. For example, features `Driver` and `Algorithms`, ii) *Optional features* may or may not be selected if their respective parent feature is selected. An example is feature `Search`,

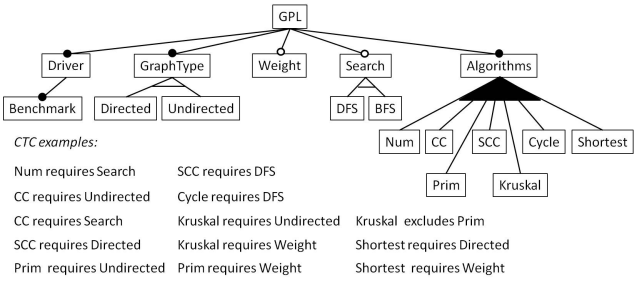

Fig. 1: Graph Product Line Feature Model

iii) *Exclusive-or relations* indicate that exactly one of the features in the exclusive-or group must be selected whenever the parent feature is selected. They are depicted as empty arcs crossing over a set of lines connecting a parent feature with its child features. For instance, if feature `Search` is selected, then either feature `DFS` or feature `BFS` must be selected, iv) *Inclusive-or relations* indicate that at least one of the features in the inclusive-or group must be selected if the parent is selected. They are depicted as filled arcs crossing over a set of lines connecting a parent feature with its child features. As an example, when feature `Algorithms` is selected then at least one of the features `Num`, `CC`, `SCC`, `Cycle`, `Shortest`, `Prim`, and `Kruskal` must be selected.

Besides the parent-child relations, features can also relate across different branches of the feature model with the so called *Cross-Tree Constraints (CTC)*. Figure 1 shows the CTCs of our feature model in textual form. For instance, `Num requires Search` means that whenever feature `Num` is selected, feature `Search` must also be selected. These constraints as well as those implied by the hierarchical relations between features are usually expressed and checked using propositional logic, for further details refer to [27].

B. Combinatorial Interaction Testing in SPLs

Combinatorial Interaction Testing (CIT) is a testing approach that constructs samples to drive the systematic testing of software system configurations [28], [6]. When applied to SPL testing, the idea is to select a representative subset of products where interaction errors are more likely to occur rather than testing the complete product family [28]. In this section we provide the basic terminology of CIT for SPLs[2].

Definition 1: **Feature list**. A feature list (`FL`) is the list of features in a feature model.

Definition 2: **Feature set**. A feature set `fs` is a 2-tuple `[sel,`$\overline{sel}$`]` where `fs.sel` and `fs.`$\overline{sel}$ are respectively the set of selected and not-selected features of a SPL product. Let `FL` be a feature list, thus `sel,`$\overline{sel} \subseteq$`FL`, `sel`$\cap\overline{sel} = \emptyset$, and `sel`$\cup\overline{sel}$`=FL`. Wherever unambiguous we use the term **product** as a synonym of feature set.

Definition 3: **Valid feature set**. A feature set `fs` is valid with respect to a feature model `fm` iff `fs.sel` and `fs.`$\overline{sel}$ do not violate any constraints described by `fm`. The set of all valid feature sets represented by `fm` is denoted as $\mathcal{FS}^{fm}$.

[2]Definitions based on [27] and [12].

GPL has 73 distinct feature sets. An example of valid feature set `fs1` is one that computes the algorithm `Number`, on `Directed` graphs using `BFS` search. Thus, the selected features are `fs1.sel={GPL, Driver, Benchmark, GraphType, Directed, Search, BFS, Algorithms, Number}`[3]. Consider now another feature set `fs2` with selected features `BFS` and `Cycle`, meaning {`BFS,Cycle`} $\subseteq$ `fs2.sel`. This feature set is invalid because these two features violate the CTC that establishes that whenever `Cycle` feature is selected then feature `DFS` must be selected, i.e. `Cycle requires DFS`.

The focus of our paper is pairwise testing, thus our concern is on the combinations between two features. The coming definitions are consequently presented with that perspective; however, the generalization to combinations of any number of features is straightforward.

Definition 4: **Pair**. A pair `ps` is a 2-tuple `[sel,`$\overline{sel}$`]` involving two features from a feature list `FL`, that is, $ps.sel \cup ps.\overline{sel} \subseteq FL \wedge ps.sel \cap ps.\overline{sel} = \emptyset \wedge |ps.sel \cup ps.\overline{sel}| = 2$. We say pair `ps` is `covered` by feature set `fs` iff $ps.sel \subseteq fs.sel \wedge ps.\overline{sel} \subseteq fs.\overline{sel}$.

Definition 5: **Valid pair**. A pair `ps` is valid in a feature model `fm` if there exists a valid feature set `fs` that covers `ps`. The set of all valid pairs of a feature model `fm` is denoted with $\mathcal{VPS}^{fm}$.

Let us illustrate pairwise testing with GPL. Some examples of pairs are: `GPL` and `Search` selected, `Weight` and `Undirected` not selected, `CC` not selected and `Driver` selected. An example of invalid pair, i.e. not denoted by the feature model, is features `Directed` and `Undirected` both selected. Notice that this pair is not valid because they are part of an exclusive-or relation.

Definition 6: **Pairwise test suite**. A pairwise test suite `pts` for a feature model `fm` is a set of valid feature sets of `fm`. A pairwise test suite is complete if it covers all the valid pairs in $\mathcal{VPS}^{fm}$, that is:
$\{\text{fs}|\forall ps \in \mathcal{VPS}^{fm} \Rightarrow \exists fs \in \mathcal{FS}^{fm}$ such that fs covers ps$\}$.

In GPL there is a total of 418 valid pairs, so a complete pairwise test suite for GPL must have all these pairs covered by at least one feature set. Henceforth, because of our focus and for sake of brevity we will refer to pairwise test suites simply as *test suites*.

C. Multi-Objective Optimization

There exists a wealth of literature in the context of Evolutionary Multi-Objective Optimization [29] and the application of *Search-Based Software Engineering (SBSE)* to software testing [30]. Our work is the first to cast CIT SPL pairwise testing as a multi-objective optimization problem and use multi-objective classical algorithms. Our definitions are based on [31], [32], [33], which we adapt to our context of pairwise testing for two objectives.

Definition 7: **Decision space**. The decision space is the set of possible solutions to an optimization problem. In our context, it corresponds to the set of all possible sets of valid feature sets represented by a feature model `fm`, denoted as $\mathcal{DS}^{fm} = \mathcal{P}(\mathcal{FS}^{fm})$. A `decision vector` is an element of the decision space, that is $x \in \mathcal{DS}^{fm}$.

Definition 8: **Objective functions**. An objective function is a function that represents a goal to optimize. In our context we consider two functions:

- Coverage function. Recall that we want to maximize the number of pairs covered by a test suite. For simplicity, we use the alternative of minimizing the number of pairs not covered which is defined as follows:
  $f_1^{fm} : \mathcal{DS}^{fm} \rightarrow \mathbb{N}$,
  $f_1^{fm}(x) = |\mathcal{VPS}^{fm} \backslash covers(x)|$,
  where `covers` computes the pairs covered by the feature sets of test suite `x`.
- Test suite size function. Recall that we want to minimize the number of feature sets in the test suite. We define this function as follows:
  $f_2^{fm} : \mathcal{DS}^{fm} \rightarrow \mathbb{N}$,
  $f_2^{fm}(x) = |x|$.

Definition 9: **Vector function**. A vector function associated to a feature model `fm` is defined as[4]:
$F^{fm} : \mathcal{DS}^{fm} \rightarrow \mathcal{OS}^{fm}$
$F^{fm}(x) = (f_1^{fm}(x), f_2^{fm}(x))$
where $\mathcal{OS}$ is the corresponding `objective space` which in our context is $\mathcal{OS}^{fm} = \mathbb{N} \times \mathbb{N}$.

Definition 10: **Objective vector**. An objective vector is the result of applying the vector function to an element of the decision space. Let `x` $\in \mathcal{DS}^{fm}$, its objective vector `u` is defined as: $u = F^{fm}(x) = (f_1^{fm}(x), f_2^{fm}(x))$.

Pareto dominance is the most commonly accepted notion of superiority in multi-objective optimization because it is the canonical generalization of the single-objective case [31].

Definition 11: **Pareto dominance**. Let $x, y \in \mathcal{DS}^{fm}$, $u = F^{fm}(x) = (u_1, u_2)$, and $v = F^{fm}(y) = (v_1, v_2)$ for a feature model `fm`. We say that objective vector `u` `Pareto-dominates` objective vector `v` iff $u \preccurlyeq v$ and $v \not\preccurlyeq u$. Intuitively $u \preccurlyeq v$ means that u is better than v if there is at least one objective i for which $f_i^{fm}(x)$ is better than $f_i^{fm}(y)$, and there are no objectives for which it is worse. More formally, for the two functions we want to minimize in our context: $u \preccurlyeq v$ iff $(u_1 < v_1 \wedge u_2 < v_2)$ or $(u_1 < v_1 \wedge u_2 = v_2)$ or $(u_1 = v_1 \wedge u_2 < v_2)$.

Definition 12: **Multi-Objective SPL pairwise testing problem**. A multi-objective pairwise SPL testing problem for a feature model `fm` is a 4-tuple $(\mathcal{DS}^{fm}, \mathcal{OS}^{fm}, F^{fm}, \preccurlyeq)$ whose goal is to find a decision vector $x^* \in \mathcal{DS}^{fm}$ such that it minimizes vector function F^{fm}.

Definition 13: **Pareto optimal decision vector**. A decision vector `x` $\in \mathcal{DS}^{fm}$ is Pareto optimal iff it does not exist another `y` $\in \mathcal{DS}^{fm}$ such that Pareto-dominates it, that is $F(y) \preccurlyeq F(x)$.

Definition 14: **Pareto optimal set**. The Pareto optimal set P_*^{fm} of a multi-objective pairwise SPL testing problem for

[3]Unselected features omitted for brevity.

[4]For notational brevity we omit on the vector function and the objective vectors the `T` that denotes the transpose on vectors.

feature model fm and its vector function F^{fm} is:
$P_*^{fm} = \{x \in \mathcal{DS}^{fm} | \neg \exists x' \in \mathcal{DS}^{fm} such\ that\ F(x') \preccurlyeq F(X)\}$.

Definition 15: **Pareto front**. For a given multi-objective pairwise SPL testing problem for feature model fm and a Pareto optimal set P_*^{fm}, the Pareto front is defined as: $PF_*^{fm} = F^{fm}(P_*^{fm})$.

III. Multi-objective Algorithms

In this section we describe the four multi-objective algorithms our analysis compares and sketch their representation and operators used.

A. Algorithms Analyzed

We selected the following four algorithms because they have been extensively and successfully applied to a large number of problem domains and represent a diverse set of techniques and concepts of multi-objective evolutionary algorithms:

- Non-dominated Sorting Genetic Algorithm (NSGA-II), builds a population of individuals which are ranked and sorted according to nondomination level. It was proposed in [19], and has become a reference algorithm in multi-objective optimization.
- Multi-Objective Cellular Genetic Algorithm (MOCell) introduced by Nebro *et al*. [20], is a cellular genetic algorithm (cGA). In cGAs, the concept of (small) *neighbourhood* is paramount. This means that an individual may only cooperate with its nearby neighbours in the breeding loop. Some studies have shown MOCell to outperform NSGA-II in concrete domains e.g. [34], [20].
- Strength Pareto Evolutionary Algorithm (SPEA2)was proposed by Zitler *et al.* in [21]. SPEA2 uses a population and an archive simultaneously in its operation.
- Pareto Archived Evolution Strategy (PAES) is a metaheuristic proposed in [22] by Knowles and Corne. The algorithm is based on a simple (1+1) evolution strategy. To find diverse solutions in the Pareto optimal set, PAES uses an external archive of non-dominated solutions, which is also used to make decisions about new candidate solutions.

B. Notes on Representation and Operators

The multi-objective algorithms have been implemented using jMetal [35], a Java framework aimed at the development, experimentation, and study of metaheuristics for solving multi-objective optimization problems. All algorithms use the same representation for an individual, which is a set of products, and also the same variation operators. We have used *binary tournament* as the selection scheme. This operator works by randomly choosing two individuals from the population and the one dominating the other is selected; if both solutions are non-dominated one of them is randomly selected. The crossover operator, which is executed with probability 0.8, takes two solutions, S_1 and S_2, then one cross-point is randomly selected in both solutions generating two parts per solution S_{1a} - S_{1b} and S_{2a} - S_{2b}. Finally, two new individuals are created $S_{1'}(S_{1a} - S_{2b})$ and $S_{2'}(S_{2a} - S_{1b})$. The mutation operator is executed with probability 0.1. It generates ten valid products, then the product that adds more coverage to the solution is added. The product that adds more coverage to the solution is known because we apply the mutation before applying the recombination. This order of the variation operators allow us to take advantage of the information collected in the evaluation of the individual. If the resulting individual has the same coverage and more test products, at the end of the iteration, the algorithms delete it from the population because this solution is dominated.

IV. Seeding Strategies

The impact that seeding strategies have on the performance of evolutionary algorithms has been documented for instance in Paul et al. [36]. Recall that the ultimate goal of a seeding strategy is to embed domain knowledge into the individuals of the population such that this knowledge is exploited when searching for solutions. Consequently, the strategies are also domain dependent. There are seeding strategies for engineering problems [37], positioning problems [38], timetabling problems [39], to cite a few examples.

In the field of software testing, seeding has also been used. For example, Alba and Chicano [40] used seeding in order to provide some individuals that were able to execute the predicates of the branches they should cover. In a recent work, Fraser and Arcuri [23] compare different seeding strategies and conclude that an improvement in the performance is achieved with statistical significance when a seeding strategies is used. It should be noted though that none of these pieces of work addresses seeding in the realm of SPL testing. To the best of our knowledge, our work is the first to explore this issue. Our focus is on measuring the impact of seeding the initial population (the first solution in PAES) of the four classical algorithms described in Section III. Next we explain the three seeding strategies that we study.

A. Size-Based Random Seeding

The *size-based random seeding strategy (SB)* leverages the knowledge of the size of known complete test suites. In our case, we used solutions generated by the algorithm CASA [9], at the core of the third seeding strategy which we explain shortly in Section IV-C. We must stress that this strategy uses from the known complete test suites *only* their *size*. We summarize this strategy with the following code snippet, where fm is a feature model and n is the size of the population to seed:

$$seed := CASA(fm)$$
$$population := sizeBasedRandom(\underline{size(seed)}, n, fm)$$

Algorithm 1 sketches how this strategy works. It receives as input the size of the test suite to generate as seed (the number of features sets of a known complete test suite), the size of the population to generate, and a feature model. The core of the algorithm is a loop (Lines 5-8) that constructs a seed test suite (of the size of a known test suite) by randomly selecting valid feature sets from a feature model (Line 6). Once the seed test suite is computed, it is passed to algorithm

`seedPopulation` (Line 9), sketched on Algorithm 2, to generate a population of the desired size which is returned.

Algorithm 1 Size-Based Random Seeding Strategy.

```
1: proc sizeBasedRandom
2: Input: seedSize:int, populationSize:int, fm:feature model
3: Output: population: set of test suite
4: seed← ∅
5: for i ← 1 ... seedSize do
6:     featureSet←RandomFeatureSets(fm)
7:     seed←seed ∪ featureSet
8: end for
9: population←seedPopulation(seed, populationSize)
10: return population
```

Algorithm 2 uses the `seed` test suite that it receives as input to generate a population of a given size. It creates the new test suites by randomly removing feature sets from the population `seed` (Line 6). This algorithm is also used by the other two seeding strategies.

Algorithm 2 Seed Population.

```
1: proc seedPopulation
2: Input: seed:test suite, populationSize:int
3: Output: population:set of test suite
4: population← ∅
5: for i ← 1 ... populationSize do
6:     testSuite← RemoveFeatureSets(seed)
7:     population← population ∪ testSuite
8: end for
9: return population
```

B. Greedy Seeding

The *greedy seeding strategy (GS)*, Algorithm 3, generates of a single `seed` test suite with complete coverage from which it seeds a population. Thus, this strategy leverages the knowledge of the feature sets that are part of a complete test suite. To create a seed, we use a constructive approach that selects on each iteration the best feature set (Line 9) out of 100 randomly generated ones (Lines 7-10) until complete coverage is reached (Line 6). Once the `seed` test suite is computed, it is passed to algorithm `seedPopulation` (Line 13), sketched on Algorithm 2, to generate a population of the desired size which is then returned. We summarize this strategy with the following code snippet, where `fm` is a feature model and `n` is the size of the population to seed:

$$population := greedySeeding(n, fm)$$

Algorithm 3 Greedy Seeding Strategy.

```
1: proc greedySeeding
2: Input: populationSize:int, fm:feature model
3: Output: population:set of test suite
4: seed← ∅
5: bestFS← ∅
6: while not Total_Coverage(seed) do
7:     for i ← 1 100 do
8:         newFS←RandomFeatureSets(fm)
9:         bestFS← ChooseBest(bestFS, newFS)
10:    end for
11:    seed← seed ∪ bestFS
12: end while
13: population←seedPopulation(seed, populationSize)
14: return population
```

C. Single-objective Based Seeding

The *single-objective based seeding strategy (SO)* consists in using a complete test suite computed by a single-objective algorithm to seed a population. Thus, this strategy leverages knowledge of test suites computed by existing single-objective SPL testing approaches. For the task of generating test suites we chose CASA, a simulated annealing algorithm that was designed to generate n-wise covering arrays for SPLs [9]. CASA relies on three nested search strategies. The outermost search performs one-sided narrowing, pruning the potential size of the test suite to be generated by only decreasing the upper bound. The mid-level search performs a binary search for the test suite size. The innermost search strategy is the actual simulated annealing procedure, which tries to find a pairwise test suite of size `N` for feature model `FM`. We selected CASA because in our previous work it performed better than other techniques and it is well-known in both search-based and SPL research communities. For more details on CASA and its comparison with other approaches please refer to [9], [41].

First, we performed 30 independent runs per feature model because CASA is a non-deterministic algorithm. Then, we randomly chose one of the solutions for seeding the population. Do notice that we used the same solution of CASA for seeding all the multi-objective algorithms in order to make a fair comparison among them. We summarize this strategy with the following code snippet, where `fm` is a feature model and `n` is the size of the population to seed:

$$population := seedPopulation(\underline{CASA(fm)}, n)$$

V. Evaluation

In this section we describe how the evaluation was carried out, the quality indicators used, the description of the study corpus, and the results obtained and analysis performed.

A. Feature Models Corpus

The experimental corpus of our evaluation is formed with 19 feature models. These models have two important characteristics to make our results directly applicable by software engineers. First, that the feature models are associated to actual SPLs whose code is publicly available or can be made accessible by request to the corresponding authors. This is important so that testing can be carried out on the SPL code. Second, that the feature models are explicitly and directly provided by the SPL authors, rather than, for instance, reverse-engineered from other artefacts. This is important because it guarantees that all the feature combinations are correctly captured in the feature models. We searched into three main SPL related websites: SPL Conqueror [42], FeatureHouse [43], and SPL2go [44]. In addition, we looked at recently published articles within the SPL community. For the management and analysis of feature models, we relied on three frameworks: SPLAR [45], FAMA [46], and SPLCA [47]. These tools in turn, imposed additional constraints to the selection of our corpus[5]. Table I summarizes the feature models used in our evaluation. It shows the number of features, number of products, and their application domain with the source where

[5]For example, the type of CTCs that FAMA can analyze.

TABLE I: Feature Models Summary

Feature Model	NF	NP	PF	Domain
Apache	10	256	6	web server [42]
argo-uml-spl	11	192	6	UML tool [49]
BDB*	117	32	NA	database [43]
BDBFootprint	9	256	6	database [42]
BDBMemory	19	3,840	NA	database [42]
BDBPerformance	27	1,440	NA	database [42]
Curl	14	1024	NA	data trasfer [42]
DesktopSearcher	22	462	8	file search [44]
fame_dbms_fm	20	320	6	database [44]
gpl	18	73	12	graph algorithms [26]
LinkedList	27	1,344	NA	data structures [42]
LLVM	12	1,024	NA	compiler library [42]
PKJab	12	72	6	messenger [42]
Prevayler	6	32	6	object persistence [42]
SensorNetwork	27	16,704	NA	networking [42]
TankWar	37	1,741,824	NA	game [43]
Wget	17	8,192	NA	file retrieval [42]
x264	17	2,048	NA	video encoding [42]
ZipMe	8	64	6	data compression [42]

NF: Number of Features, **NP**:Number of Products,
PF: Pareto Front size, NA if not available.
*__BDB__ prefix stands for Berkeley database.

we obtained them from. Additionally, we provide the size of the Pareto front[6], if known.

B. Experimental Setting

This section describes how the evaluation was carried out. The four algorithms we analyzed are heuristic, thus we performed 30 independent runs for a meaningful statistical analysis. In order to measure the performance of the multi-objective algorithms used here, the quality of their resulting nondominated set of solutions has to be considered. In addition to the two objective functions defined in Section II-C, coverage and products, we selected two well-known quality indicators that are commonly used in the multi-objective community to compare the approximated Pareto fronts of several algorithms[7] Hypervolume (HV) [24] and Generational Distance (GD) [25]. In addition to these quality measures, we have also analyzed the time required to run the multi-objective algorithms and obtain the Pareto front, since we want the algorithms to be as fast as possible. All the executions were run in a cluster of 16 machines with Intel Core2 Quad processors Q9400 (4 cores per processor) at 2.66 GHz and 4 GB memory running Ubuntu 12.04.1 LTS and managed by the HT Condor 7.8.4 cluster manager. Since we have 4 algorithms, 3 different seeding strategies, and 19 feature models the total number of independent runs is $4 \times 3 \times 19 \times 30 = 6,840$. The stopping criterion was 1,000 evaluations. The sources and data of our work are available at [41].

C. Experimental Results

This section presents the statistical analysis performed and the results obtained.

1) Wilcoxon Test: In order to check if the differences between the algorithms are statistically significant or just a matter of chance, we applied the non-parametric Wilcoxon rank-sum test [50]. The confidence level used is 95% (*p*-value under 0.05).

[6]Computed with algorithm presented in [48].

[7]We used a reference front whenever the true Pareto front was unknown.

We first analyze the behaviour of the multi-objective algorithms with the aim of highlighting which algorithm works better. In Table II we show the average values of the quality indicators grouped by algorithm and the execution time. Regarding the significant differences, they exist only between PAES and the rest of the algorithms for both quality indicators and performance measure. We must point out that even though PAES seems to be the worst algorithm, the average generational distance value obtained by PAES is better than the obtained by NSGA-II. This high value of generational distance requires a further analysis (addressed shortly) because NSGA-II obtains the best values for the rest of the indicators, HV and time. SPEA2 is the best in generational distance. Our evaluation indicates there is not an algorithm which is the best for all quality indicators, nonetheless NSGA-II performs best in 2 out 3 indicators.

TABLE II: Comparison of multi-objective algorithms using the proposed quality indicators and performance time.

Algorithms	HV	GD	Time(ms)
NSGA-II	0.6583	0.0396	70523
MOCell	0.6553	0.0293	74325
SPEA2	0.6533	0.0289	71349
PAES	0.6390	0.0351	101246

In Table III we summarize the average results obtained grouped by seeding strategy. In this case there are clear significant differences among all the seeding strategies. We have highlighted the best value per quality indicator and performance measure, which is always obtained with the SO seeding strategy. First, we analyze the hypervolume, the higher the value, the better the quality of the obtained results. The best value is obtained with the SO seeding strategy. There are statistical significant differences between SB and GS, and also between SB and SO. Thus, it can be concluded that the quality of solutions obtained with SB are worse than those obtained with the other strategies. Second, we analyze the generational distance, the lower the value, the better the quality of the results. The best value is obtained with the SO seeding strategy again. In addition, there are significant differences with the other seeding strategies for the generational distance indicator. Finally, we analyze the time spent in the generation of the Pareto front. The results are clear, the SO strategy is the fastest. Recall that the execution of the CASA algorithm for seeding is included, which on average is 2905 milliseconds.

TABLE III: Comparison of seeding strategies using the proposed quality indicators and performance time.

Seeding Strategy	HV	GD	Time(ms)
SizeBased Random (SB)	0.6421	0.0427	138404
Greedy (GS)	0.6556	0.0447	76783
SingleObjective (SO)	0.6568	0.0123	25800

2) $\hat{A}_{12}$ Statistic: Statistical difference has been measured with the Wilcoxon test. In order to properly interpret the results of statistical tests, it is always advisable to report effect size measures. For that purpose, we have also used the non-parametric effect size measure $\hat{A}_{12}$ statistic proposed by Vargha and Delaney [51], as recommended by Arcuri and Briand [52]. Given a performance measure M, $\hat{A}_{12}$ measures the probability that running algorithm A yields higher M

values than running another algorithm *B*. If the two algorithms are equivalent, then $\hat{A}_{12} = 0.5$. If $\hat{A}_{12} = 0.3$ entails one would obtain higher values for M with algorithm *A*, 30% of the time.

Table IV shows the $\hat{A}_{12}$ statistic to assess the practical significance of the results. We have highlighted the largest distance from 0.5 (equality) per quality indicator, note that 0.5 indicates no difference in the comparison. Regarding HV, there are no big differences between the algorithms. The highest difference occurs between NSGA-II and PAES. Regarding GD, the highest difference is between MOCell and PAES. There, the generational distance is larger in 41.94% of the times. In addition, notice that NSGA-II obtains smaller values than PAES with probability 0.544 (1-0.4560). This result indicates a tendency contrary to the deduced from the previous average value of GD of NSGA-II, in most of the comparisons it achieves a lower value than PAES. Regarding time, NSGA-II is faster with more probability than the other algorithms. In general, this statistic confirms again that NSGA-II obtains better Pareto fronts than the other algorithms, according to the selected quality indicators, and faster.

TABLE IV: $\hat{A}_{12}$ statistical test results for all algorithms. NSGA-II yields better results for HV and time measures.

Algorithms	HV	GD	Time
NSGAII-SPEA2	0.5182	0.5172	0.4904
NSGAII-MOCell	0.5112	0.5202	0.4816
NSGAII-PAES	0.5626	0.4560	0.2839
SPEA2-MOCell	0.4932	0.5039	0.4910
SPEA2-PAES	0.5447	0.4205	0.3019
MOCell-PAES	0.5521	0.4194	0.3027

We now comment on the $\hat{A}_{12}$ statistic values shown in Table V that compares the seeding strategies. The SO strategy obtains higher values of HV than the other strategies, it obtains better values of HV than SB and GS strategies in a 54.42% (probability 1-0.4558) and a 50.23% (probability 1-0.4977), respectively. For GD, the SO strategy obtains better values (lower values) than SB and GS in a 85.62% and in a 78.39%, respectively. Therefore, SO is widely best for the generational distance indicator. Finally, SO spends less time than SB and GS in 86.19% and 82.27%, respectively. Thus, the SO strategy is the fastest without any doubt.

TABLE V: $\hat{A}_{12}$ statistical test results for seeding strategies. SO yields better quality indicators and time values.

Seeding Strategy	HV	GD	Time
SizedBased(SB) – Greedy(GS)	0.4568	0.4795	0.6377
SizedBased(SB) – SingleObjective(SO)	0.4558	0.8562	0.8619
Greedy(GS) – SingleObjective(SO)	0.4977	0.7839	0.8227

D. Analysis

With the results presented in the previous section we are now able to answer our research questions. Regarding **RQ1**, we didn't observe in our results a clear winner algorithm. We can claim, however, that PAES seems to provide approximated Pareto fronts of lower quality than the ones of the other algorithms. PAES is a trajectory-based algorithm (works with one solution and not a population), and probably this makes it less competitive in this problem, in which having a diverse population seems to be beneficial.

From the point of view of the testing engineer that faces this problem, our recommendation is to use any of the three best algorithms, NSGA-II, SPEA2 or MOCell, with the SO seeding strategy. The results obtained are approximated Pareto fronts like the one in Figure 2. Each point in that front represents a test suite for the SPL, that is, a set of products. The number of products and the coverage are given by the coordinates of the point. Thus, the multi-objective approach offers the engineer a set of test suites of different size having (almost) optimal coverage. The engineer can, then, select the most appropriate test suite depending on the resources s/he has to test the SPL or the level of coverage s/he has to satisfy. It is important to note that this selection of the most appropriate test suite can be done after the algorithms did their job. This contrasts with the scalarization technique used in the related work (i.e. [16] and [17]) that transform the multi-objective problem in a single-objective one. In that case, only one solution is provided and the testing engineer has to provide a weight for each objective function to indicate the relative importance of the objectives. Thus, s/he is somehow selecting the solution beforehand, without the possibility of taking a look to all other, probably interesting, solutions.

Regarding **RQ2**, we can say that the seeding strategy has an impact on the performance of the search algorithm. Seeding the multi-objective algorithm with good quality solutions obtained using CASA reduces the time required in the search and finds the approximated Pareto front that is nearer to the Pareto front. These initial "seeds" include some desirable building blocks (i.e. feature sets) of the optimal solutions that help the algorithm in its search.

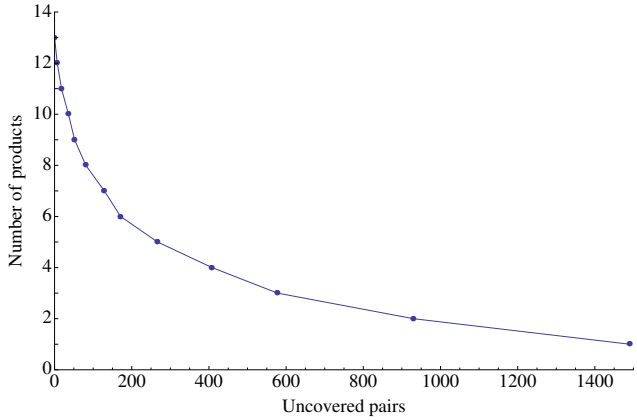

Fig. 2: Approximated Pareto front obtained by NSGA-II in TankWar.

E. Threats to Validity

We have taken special care to address the validity threats highlighted by Ali et al. [53] that are applicable to our work. Adequate parameter setting is a common internal validity threat. In this paper we used standard parameter values for the four algorithms we analyzed. Extensive evidence provided by the work of Arcuri and Fraser [54] suggest that default values might be good enough for assessing some search based testing techniques. However, how their findings map to the context of SPL pairwise testing is an open question that we plan to address in our future work.

We identified two external threats: the selection of the feature models corpus and the selection of the multi-objective algorithms. To address the first of these threats, we searched on all the website repositories of the SPL community and conferences with SPL related topics for examples of feature models with the characteristics mentioned in Section V-A and include as many as possible within the feasibility of the current tool support. Certainly other case studies may emerge that would be worth including and might yield different results. Currently our next target is to analyze the so-called *feature models in the wild* [55]. This set of case studies are perhaps the largest and most complex collection of feature models, most of them stem from the operating systems domain. Coping with their level of complexity does demand significant tool improvement beyond the scope of the current work. For instance, improvements are needed to effectively represent larger individuals and to efficiently implement the required evolutionary operators in combination with the underlying feature modelling reasoning tools. To address the second external threat, we included four distinct algorithms that represent a diverse array of multi-objective techniques and concepts. Certainly, there might be other algorithms that could potentially yield better results. We plan to explore other alternatives (i.e. [31], [32], [35]) as part of our future work.

VI. Related Work

Recent surveys and mapping studies on SPL testing [4], [5], attest not only the increasing relevance of the topic within the SPL community but also confirm that exploiting multi-objective optimization approaches for SPL testing remains an area largely unexplored. In this section we briefly summarize those pieces of work closest to ours.

Multi-Objective Optimization for SPL. The work by Wang et al. present an approach to minimize test suites using weights in the fitness function [17], that is, it uses a *scalarizing function* that transforms a multi-objective problem to a single-objective one [33]. Their work uses three objectives: test minimization percentage, pairwise coverage, and fault detection capability. Recent work by Henard et al. [16] presents an ad-hoc multi-objective algorithm whose fitness function is also scalarized. Their work focuses also on maximizing coverage, minimizing test suite size, and minimizing cost. They perform a basic comparison against random solutions for 8 feature models.

Undoubtedly, there is a clear and sharp contrast with our work. First, none of these pieces of work analyses classical multi-objective evolutionary algorithms. Second, both approaches use scalarizing functions so they are not pure multi-objective algorithms. Incidentally we should point out there is an extensive body of work on the downsides of scalarization in multi-objective optimization [18]. Among the shortcomings are the fact that weights may show a preference of one objective over the other and, most importantly, the impossibility of reaching some parts of the Pareto front when dealing with convex fronts. Third, their evaluation does not consider the impact of seeding. Fourth, they do not employ well-established and well-accepted multi-objective quality indicators in their comparisons. Fifth, the number and diversity (i.e. number of features, number of products, domain) of the feature models they analysed is more restricted than our feature model corpus[8].

Work by Lopez-Herrejon et al. proposed an exact algorithm that computes the true Pareto fronts of a feature model using SAT solvers [48]. However, this approach suffers from scalability issues. The Pareto front could be computed only 10 out of the 19 feature models studied in this work. In contrasts, this paper studies the use classical multi-objective algorithms to address this limitation. We believe both approaches could be complementary, like we did for instance use the true Pareto front to compute the quality indicators.

Search-Based Software Testing for SPL. *Search-Based Software Engineering (SBSE)* focuses on the application of search-based optimization techniques to problems in software engineering [56]. Among the techniques SBSE relies on are evolutionary computation techniques and basic searches such as hill climbing or simulated annealing. Within SBSE a major research focus has been software testing [56]. A recent overview by McMinn highlights the major achievements made in Search-Based Software Testing and some of the open questions and challenges [57]. Except for a few efforts, summarized next, the application of search based techniques to SPL testing remains largely unexplored. Garvin et al. applied simulated annealing to combinatorial interaction testing for computing `n`-wise coverage for SPLs [9]. Ensan et al. propose a genetic algorithm approach for test case generation for SPLs that uses as fitness function a variation of cyclomatic complexity metric adapted to feature models [8]. Henard et al. propose an approach that uses a dissimilarity metric that favours individuals whose n-wise coverage varies the most from the current population thus increasing the chances of wider coverage [10]. In clear contrast with our work, these last two approaches are single-objective. Recent work by Xu et al. uses a genetic algorithm for continuous test augmentation [15]. Their CONTESA tool incrementally generates test cases employing static analysis techniques for achieving coverage more effectively. It is in our future work to study how to leverage such static analysis techniques and use them as other objectives to consider in the multi-objective optimization.

Combinatorial Testing for SPL. Perrouin et al. propose an approach that translates t-wise coverage problems into Alloy programs and rely on its automatic instance generation to obtain covering arrays [14]. Oster et al. propose MoSo-PoLiTe [13], an approach that translates feature models and their constraints into binary constraint solver problems (CSP) from which they compute pairwise covering arrays. MoSo-PoLiTe can also include pre-selected products as part of the covering arrays. These two approaches were subsequently analysed by Perrouin et al. with a framework they propose to compare t-wise SPL testing [58]. Similarly, Hervieu et al. developed a tool called PACOGEN that also relies on constraint programming for computing pairwise coverage from feature models [11]. This tool has been recently included as part of a framework for practical pairwise testing in industrial settings [59]. Johansen et. al propose a greedy approach to generate n-wise test suites that adapts of Chvátal's algorithm to solve the set cover problem [12]. None of these approaches use multi-objective evolutionary algorithms.

[8]We do not to include the feature models of those papers because, to the best of our knowledge, they do not have the characteristics described in Section V-A.

VII. Conclusions and Future Work

This paper is the first study of classical multi-objective evolutionary techniques applied to SPL pairwise testing. The group of algorithms were selected to cover a diverse array of techniques and concepts of multi-objective evolutionary computing. In addition, we study the impact of seeding in performance. Our evaluation unequivocally showed that seeding with knowledge from a single-objective technique produces significantly better results in less time. It also suggests that using this seeding strategy with either of NSGA-II, SPEA2 or MOCell yields results of similar quality. Our findings enable software engineers facing SPL combinatorial testing challenges to select not just one solution (as in the case of single-objective techniques) but instead to select from an array of test suite possibilities that can better match their economical or technological constraints.

Our work opens up several research venues which we plan to address as part of our future work. As mentioned before, we plan to extend the feature model corpus and the group of multi-objective algorithms analyzed and study the impact of parameter setting. Besides these objectives, our goal is to expand our study beyond pairwise, to integrate domain knowledge such as control flow information as an optimization objective for test code generation, and to characterize when a particular multi-objective algorithm performs better based, for example, on structural metrics of the feature models [60].

Acknowledgements

Funded by Austrian Science Fund (FWF) project P25289-N15 and Lise Meitner Fellowship M1421-N15, the Spanish Ministry of Economy and Competitiveness and FEDER under contract TIN2011-28194 and fellowship BES-2012-055967. It is also partially founded by project 8.06/5.47.4142 in collaboration with the VSB-Tech. Univ. of Ostrava and Universidad de Málaga, Andalucía Tech.

References

[1] P. Zave, "Faq sheet on feature interaction," http://www.research.att.com/ pamela/faq.html.

[2] K. Pohl, G. Bockle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-21, 1990.

[4] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *Information & Software Technology*, vol. 53, no. 5, pp. 407–423, 2011.

[5] E. Engström and P. Runeson, "Software product line testing - a systematic mapping study," *Information & Software Technology*, vol. 53, no. 1, pp. 2–13, 2011.

[6] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, Feb. 2011. [Online]. Available: http://doi.acm.org/10.1145/1883612.1883618

[7] H. Cichos, S. Oster, M. Lochau, and A. Schürr, "Model-based coverage-driven test suite generation for software product lines," in *MoDELS*, ser. Lecture Notes in Computer Science, J. Whittle, T. Clark, and T. Kühne, Eds., vol. 6981. Springer, 2011, pp. 425–439.

[8] F. Ensan, E. Bagheri, and D. Gasevic, "Evolutionary search-based test generation for software product line feature models," in *CAiSE*, ser. Lecture Notes in Computer Science, J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza, Eds., vol. 7328. Springer, 2012, pp. 613–628.

[9] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing," *Empirical Software Engineering*, vol. 16, no. 1, pp. 61–102, 2011.

[10] 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*, vol. abs/1211.5451, 2012.

[11] A. Hervieu, B. Baudry, and A. Gotlieb, "Pacogen: Automatic generation of pairwise test configurations from feature models," in *ISSRE*, T. Dohi and B. Cukic, Eds. IEEE, 2011, pp. 120–129.

[12] M. F. Johansen, Ø. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *SPLC (1)*, E. S. de Almeida, C. Schwanninger, and D. Benavides, Eds. ACM, 2012, pp. 46–55.

[13] S. Oster, F. Markert, and P. Ritter, "Automated incremental pairwise testing of software product lines," in *SPLC*, ser. Lecture Notes in Computer Science, J. Bosch and J. Lee, Eds., vol. 6287. Springer, 2010, pp. 196–210.

[14] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. L. Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *ICST*. IEEE Computer Society, 2010, pp. 459–468.

[15] Z. Xu, M. B. Cohen, W. Motycka, and G. Rothermel, "Continuous test suite augmentation in software product lines," in *SPLC*, T. Kishi, S. Jarzabek, and S. Gnesi, Eds. ACM, 2013, pp. 52–61.

[16] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Multi-objective test generation for software product lines," in *SPLC*, T. Kishi, S. Jarzabek, and S. Gnesi, Eds. ACM, 2013, pp. 62–71.

[17] S. Wang, S. Ali, and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms," in *GECCO*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 1493–1500.

[18] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004. [Online]. Available: http://dx.doi.org/10.1007/s00158-003-0368-6

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm : {NSGA-II}," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[20] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "MOCell: A cellular genetic algorithm for multiobjective optimization," *Int. J. Intell. Syst.*, vol. 24, no. 7, pp. 726–746, 2009.

[21] E. Zitzler, M. Laumanns, and L. Thiele, "{SPEA2}: Improving the Strength Pareto Evolutionary Algorithm," Gloria\-strasse 35, CH-8092 Zurich, Switzerland, Tech. Rep. 103, 2001.

[22] P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds., *The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation*, vol. 1. Mayflower Hotel, Washington D.C., USA: IEEE Press, 1999.

[23] G. Fraser and A. Arcuri, "The seed is strong: Seeding strategies in search-based software testing," in *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, ser. ICST '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 121–130. [Online]. Available: http://dx.doi.org/10.1109/ICST.2012.92

[24] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[25] D. A. Van Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations," Ph.D. dissertation, Wright Patterson AFB, OH, USA, 1999, aAI9928483.

[26] R. E. Lopez-Herrejon and D. S. Batory, "A standard problem for evaluating product-line methodologies," in *GCSE*, ser. Lecture Notes in Computer Science, J. Bosch, Ed., vol. 2186. Springer, 2001, pp. 10–24.

[27] D. Benavides, S. Segura, and A. R. Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, 2010.

[28] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints:

A greedy approach," *IEEE Trans. Software Eng.*, vol. 34, no. 5, pp. 633–650, 2008.

[29] C. C. Coello, "Evolutionary multi-objective optimization website," http://delta.cs.cinvestav.mx/~ccoello/EMOO/.

[30] Y. Zhang, "Search based software engineering repository," http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.

[31] C. C. Coello, G. B. Lamont, and D. A. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed., ser. Genetic and Evolutionary Computation. Springer, 2007.

[32] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, 1st ed. Wiley, June 2001.

[33] E. Zitzler, "Evolutionary multiobjective optimization," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Springer, 2012, pp. 871–904.

[34] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "Design Issues in a Multiobjective Cellular Genetic Algorithm," in *Evolutionary Multi-Criterion Optimization. 4th International Conference, EMO 2007*, ser. Lecture Notes in Computer Science, S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, Eds., vol. 4403. Springer, 2007, pp. 126–140.

[35] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, "jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics," Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, Tech. Rep. ITI-2006-10, December 2006.

[36] P. V. Paul, A. Ramalingam, R. Baskaran, P. Dhavachelvan, K.Vivekanandan, R. Subramanian, and V. Venkatachalapathy, "Performance analyses on population seeding techniques for genetic algorithms," *International Journal of Engineering and Technology*, vol. 5, no. 3, pp. 2993–3000, 2013.

[37] P. Ponterosso and D. S. J. Fox, "Heuristically seeded genetic algorithms applied to truss optimisation," *Engineering with Computers*, vol. 15, no. 4, pp. 345–355, 1999. [Online]. Available: http://dx.doi.org/10.1007/s003660050029

[38] B. Saavedra-Moreno, S. Salcedo-Sanz, A. Paniagua-Tineo, L. Prieto, and A. Portilla-Figueras, "Seeding evolutionary algorithms with heuristics for optimal wind turbines positioning in wind farms," *Renewable Energy*, vol. 36, no. 11, pp. 2838 – 2844, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S096014811100190X

[39] E. K. Burke, J. P. Newall, and R. F. Weare, "Initialization strategies and diversity in evolutionary timetabling," *Evol. Comput.*, vol. 6, no. 1, pp. 81–103, Mar. 1998. [Online]. Available: http://dx.doi.org/10.1162/evco.1998.6.1.81

[40] E. Alba and F. Chicano, "Observations in using parallel and sequential evolutionary algorithms for automatic software testing," *Computers & Operations Research*, vol. 35, no. 10, pp. 3161 – 3183, 2008, part Special Issue: Search-based Software Engineering. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054807000378

[41] "Paper code and data repository," 2014, http://neo.lcc.uma.es/staff/javi/resources.html.

[42] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov, "Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption," *Information & Software Technology*, vol. 55, no. 3, pp. 491–507, 2013.

[43] "FeatureHouse website," 2013, http://www.fosd.de/fh.

[44] "SPL2go website," 2013, http://spl2go.cs.ovgu.de/.

[45] "SPLAR library," 2013, https://code.google.com/p/splar/.

[46] P. Trinidad, D. Benavides, A. Ruiz-Cortes, S. Segura, and A. Jimenez, "Fama framework," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, Sept., pp. 359–359.

[47] "PLCA Tool," 2012, http://heim.ifi.uio.no/martifag/models2012/.

[48] R. E. R. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, and E. Alba, "Multi-objective Optimal Test Suite Computation for Software Product Line Pairwise Testing," in *IEEE International Conference on Software Maintenance*, Sep. 2013, pp. 404–407.

[49] "Argouml-spl project," 2013, http://argouml-spl.tigris.org/.

[50] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC; 4 edition, 2007.

[51] A. Vargha and H. D. Delaney., "A critique and improvement of the cl common language effect size statistics of mcgraw and wong." *Journal of Educational and Behavioral Statistics*, vol. 25(2), pp. 101–132, 2000.

[52] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering." *Softw. Test. Verif. Reliab*, 2012.

[53] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Trans. Software Eng.*, vol. 36, no. 6, pp. 742–762, 2010.

[54] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[55] T. Berger, "Variability modeling in the wild," in *SPLC (2)*, E. S. de Almeida, C. Schwanninger, and D. Benavides, Eds. ACM, 2012, pp. 233–241.

[56] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, p. 11, 2012.

[57] P. McMinn, "Search-based software testing: Past, present and future," in *ICST Workshops*. IEEE Computer Society, 2011, pp. 153–163.

[58] 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*, vol. 20, no. 3-4, pp. 605–643, 2012.

[59] D. Marijan, A. Gotlieb, S. Sen, and A. Hervieu, "Practical pairwise testing for software product lines," in *SPLC*, T. Kishi, S. Jarzabek, and S. Gnesi, Eds. ACM, 2013, pp. 227–235.

[60] E. Bagheri and D. Gasevic, "Assessing the maintainability of software product line feature models using structural metrics," *Software Quality Journal*, 2010.

[61] T. Kishi, S. Jarzabek, and S. Gnesi, Eds., *17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013*. ACM, 2013.