Evolutionary Computation for Software Product Line Testing: An Overview and Open Challenges

Roberto E. Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Alexander Egyed and Enrique Alba

Abstract Because of economical, technological and marketing reasons today's software systems are more frequently being built as families where each product variant implements a different combination of features. Software families are commonly called Software Product Lines (SPLs) and over the past three decades have been the subject of extensive research and application. Among the benefits of SPLs are: increased software reuse, faster and easier product customization, and reduced time to market. However, testing SPLs is specially challenging as the number of product variants is usually large making it infeasible to test every single variant. In recent years there has been an increasing interest in applying evolutionary computation techniques for SPL testing. In this chapter, we provide a concise overview of the state of the art and practice in SPL testing with evolutionary techniques as well as to highlight open questions and areas for future research.

Keywords Software product lines · Product line testing · Search based software engineering · Feature models · Feature set · Reverse engineering · Variability modeling

---

R.E. Lopez-Herrejon (✉) · A. Egyed
Software Systems Engineering Institute, Johannes Kepler University, Linz, Austria
e-mail: roberto.lopez@jku.at

A. Egyed
e-mail: alexander.egyed@jku.at

J. Ferrer · F. Chicano · E. Alba
Universidad de Málaga, Andalucía Tech, Sevilla, Spain
e-mail: ferrer@lcc.uma.es

F. Chicano
e-mail: chicano@lcc.uma.es

E. Alba
e-mail: eat@lcc.uma.es

© Springer International Publishing Switzerland 2016

W. Pedrycz et al. (eds.), *Computational Intelligence and Quantitative Software Engineering*, Studies in Computational Intelligence 617,
DOI 10.1007/978-3-319-25964-2_4

1 Introduction

A *Software Product Line* (*SPL*) is a family of related software systems each of which provides a different combination of features [1], where a *feature* is commonly defined as an increment in program functionality [2]. Extensive research and practice attest to the substantial benefits of SPL practices such as increased software reuse, faster product customization, and reduced time to market (e.g. [3]). SPLs typically involve large number of software systems, which make it infeasible to individually test each one of them. To address this need, several testing techniques and approaches have been proposed, all with distinct advantages and drawbacks [4–7].

Search Based Software Engineering (*SBSE*) is an emerging discipline that focuses on the application of search-based optimization techniques to software engineering problems [8]. Among the techniques SBSE relies on is *evolutionary computation*—an area of artificial intelligence that studies algorithms that follow Darwinian principles of evolution [9]. Evolutionary computation techniques are generic, robust, and have been shown to scale to large search spaces. These properties have been extensively exploited for testing standard one-off systems (e.g. [10]), but their application to SPLs remains largely unexplored.

In this book chapter we present a concise overview of current techniques for SPL testing, describe and illustrate the salient work on evolutionary computation techniques applied to SPL testing, and highlight some of the open challenges that remain to be addressed. The chapter is structured as follows. Section 2 provides the basic background on SPL and evolutionary algorithms needed for this chapter. Section 3 provides a general overview on the state of the art of SPL testing. Section 4 describes *Combinatorial Interaction Testing* (*CIT*), the main approach for evolutionary SPL testing, and presents a simple illustrative algorithm that follows this approach. Section 5 presents a formal description of SPL testing as a multi-objective optimization problem, describes an algorithm to compute exact Pareto fronts, and summarizes the state of the art of research in this area. Section 7 summarizes the open questions and challenges. Section 8 presents the conclusions to our work.

2 Background

In this section we provide the basic background on the two topics that crosscut the chapter: Software Product Lines and Evolutionary Algorithms.

2.1 SPL Foundations—Feature Models and Running Example

Feature models have become a de facto standard for modelling the common and variable features of an SPL [11]. Features are depicted as labelled boxes and their relationships as lines, collectively forming a tree-like structure. Feature models then denote the set of feature combinations that the systems of an SPL can have [11, 12].

Figure 1 shows the feature model of our running example, the *Graph Product Line* (*GPL*), a standard SPL of basic graph algorithms that has been extensively used as a case study in the SPL community [13]. In this SPL, a software system has feature `GPL` (the root of the feature model) which contains its core functionality, and a driver program (`Driver`) that sets up the graph examples (`Benchmark`) to which a combination of graph algorithms (`Algorithms`) are applied. The graphs (`GraphType`) can be either directed (`Directed`) or undirected (`Undirected`), and can optionally have weights (`Weight`). Two graph traversal algorithms (`Search`) can be optionally provided: Depth First Search (`DFS`) or Breadth First Search (`BFS`). A software system must provide at least one of the following algorithms: numbering of nodes in the traversal order (`Num`), connected components (`CC`), strongly connected components (`SCC`), cycle checking (`Cycle`), shortest path (Shortest), minimum spanning trees with Prim's algorithm (`Prim`) or Kruskal's algorithm (`Kruskal`).

In a feature model, each feature has exactly one parent feature and can have a set of child features. A child feature can only be selected in a feature combination of a valid software system if its parent is selected as well. The exception is the root feature that does not have any parent and it is always selected in any software system of a SPL. There are four kinds of feature relationships:

– *Mandatory features* are depicted with a filled circle. A mandatory feature is selected whenever its respective parent feature is selected. For example, features `Algorithms` and `GraphType`.

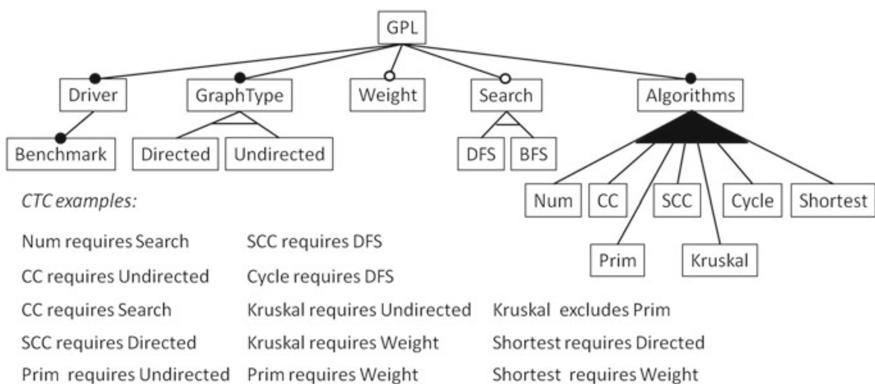

Fig. 1 Graph Product Line Feature Model [13]

- *Optional features* are depicted with an empty circle. An optional feature may or may not be selected if its respective parent feature is selected. An example is feature `Search`.
- *Exclusive-or relations* are depicted as empty arcs crossing over the lines connecting a parent feature with its child features. They indicate that *exactly one* of the features in the exclusive-or group must be selected whenever the parent feature is selected. For example, if feature `GraphType` is selected, then either feature `Directed` or feature `Undirected` must be selected.
- *Inclusive-or relations* are depicted as filled arcs crossing over a set of lines connecting a parent feature with its child features. They indicate that *at least one* of the features in the inclusive-or group must be selected if the parent is selected. If for instance, feature `Algorithms` is selected then at least one of the features `Num`, `CC`, `SCC`, `Cycle`, `Shortest`, `Prim`, or `Kruskal` must be selected.

Besides the parent-child relations, features can also relate across different branches of the feature model with *Cross-Tree Constraints* (*CTCs*). Figure 1 textually shows the CTCs of GPL. For instance, `Cycle requires DFS` means that whenever feature `Cycle` is selected, feature `DFS` must also be selected. As another example, `Prim excludes Kruskal` means that both features cannot be selected at the same time in any product. 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 [12]. Now we present the basic definitions on which SPL testing terminology is defined in the next section.

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

The `FL` for the GPL feature model is [`GPL, Driver, Benchmark, GraphType, Directed, Undirected, Weight, Search, DFS, BFS, Algorithms, Num, CC, SCC, Cycle, Shortest, Prim, Kruskal`].

Definition 2 (*Feature set*) A feature set `fs` is a 2-tuple [$\mathtt{sel}, \overline{\mathtt{sel}}$] where `fs.sel` and $\mathtt{fs}.\overline{\mathtt{sel}}$ are respectively the set of selected and not-selected features in a system part of a SPL. Let `FL` be a feature list, thus $\mathtt{sel}, \overline{\mathtt{sel}} \subseteq \mathtt{FL}$, $\mathtt{sel} \cap \overline{\mathtt{sel}} = \emptyset$, and $\mathtt{sel} \cup \overline{\mathtt{sel}} = FL$. 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 $\mathtt{fs}.\overline{\mathtt{sel}}$ do not violate any constraints described by fm. The set of all valid feature sets represented by fm is denoted as $\mathcal{FS}^{fm}$.

GPL has 73 distinct valid feature sets, some of them depicted in Table 1, where selected features are ticked (✓) and unselected features are empty. An example of valid feature set is `fs1` that computes the algorithms `Kruskal` and `CC`, on `Undirected` graphs using `DFS` search. Thus, the selected features are `fs1.sel={GPL, Driver, GraphType, Weight, Search, Algorithms, Benchmark, Undirected, DFS, CC, Kruskal}`, and the unselected features $\mathtt{fs1}.\overline{\mathtt{sel}}$`={Directed, BFS, Num, SCC, Cycle, Shortest,`

Table 1 Sample feature sets of GPL

FS	GPL	Dri	Gtp	W	Se	Alg	B	D	U	DFS	BFS	N	CC	SCC	Cyc	Sh	Prim	Kru
fs0	✓	✓	✓	✓		✓	✓		✓								✓	
fs1	✓	✓	✓	✓	✓	✓	✓		✓	✓			✓					✓
fs2	✓	✓	✓		✓	✓	✓	✓		✓		✓			✓			
fs3	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓				✓		
fs4	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		
fs5	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓		✓	
fs6	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓					✓
fs7	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓			

Driver (Dri), GraphType (Gtp), Weight (W), Search (Se), Algorithms (Alg), Benchmark (B), Directed (D), Undirected (U), Num (N), Cycle (Cyc), Shortest (Sh), Kruskal (Kr)

Prim}. Consider now another feature set gs with selected features BFS and Cycle, meaning {BFS,Cycle} $\subset$ gs.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.

2.2 Basics of Evolutionary Algorithms

Evolutionary Computation is an area of computer science, artificial intelligence more concretely, that studies algorithms that follow Darwinian principles of evolution [9]. Algorithm 1 sketches the general structure of an evolutionary algorithm adapted from [9, 14]. It starts by creating an initial population of candidate solutions for the particular problem to address (Lines 1–2). The population is denoted by term P(t) where t stands for the generation of the population. A measure of fitness to solve the problem is used to evaluate each member of the population (Line 3). Then, while not reaching a termination condition such as a given number of generations or fitness threshold (Lines 4–9), a new population is selected from the previous population and the newly created offspring (Lines 5–6). The new population is randomly mutated to promote solution diversity (Line 7) and is subsequently re-evaluated (Line 8).

Algorithm 1. Basic Evolutionary Algorithm

1: $t \leftarrow 0$
2: *initialize* $P(t)$
3: *evaluate* $P(t)$
4: **while not** *terminationCondition* **do**
5: $\quad t \leftarrow t + 1$
6: $\quad$ *select* $P(t)$ *from* $P(t-1)$
7: $\quad$ *mutate* $P(t)$
8: $\quad$ *evaluate* $P(t)$
9: **end while**

There are several types of evolutionary algorithms [9]; however, *Genetic Algorithms* (*GA*) are undoubtedly the most commonly used ones [15]. They typically employ a binary list representation and are commonly used for optimization problems such as job scheduling problems. In the coming sections we will explain how this basic algorithm is adapted for the problem of SPL testing.

3 Overview of SPL Testing

As SPL development practices become more prevalent, there is an increasing need of adequate and scalable SPL testing techniques. In recent years, there has been a growing interest by the research and practitioners communities to propose and

evaluate new methods and tools to address this need. The results have been captured and analyzed in several systematic mapping studies and surveys. In this section, we summarize the most salient works among such studies and surveys to provide the context on which to place evolutionary computation techniques for SPL.

Engström and Runerson [4] performed a mapping study that takes a higher level view of the subject by focusing for instance on the organization and process for testing, and the type of testing techniques performed such as acceptance testing, unit testing, or integration testing. A similar and complementary mapping study was carried out by Neto et al. [5]. They analyzed, for instance, the different strategies that have been taken for SPL testing such as testing product by product, incremental testing (i.e. first product tested individually and the following products with regression testing), or opportunistic reuse (i.e. employ test assets available from other products). They also analyzed others factors and aspects of SPL testing such as the use of static and dynamic analysis techniques, the effort reduction, or non-functional testing. Among their most salient findings, is the lack of evidence on when to select a given testing strategy considering factors such as development processes employed or delivery time and budget constraints. The complementary nature of these two studies has been further analyzed and reported [16, 17].

More recent studies by do Carmo Machado et al. [7, 18] have taken a closer look at the techniques used for SPL testing. They classify the works in two so-called *interests*: papers that focus on selecting the products of the SPL to test, and papers that describe approaches to actually carry out the testing on the selected products. Their study found that the *de facto* approach for selecting which SPL products to test is *Combinatorial Interaction Testing* (*CIT*) which aims at constructing *samples* to drive the systematic testing of software system configurations [19, 20]. For the second interest, they found an array of techniques, mostly based on extensions to UML activity and sequence diagrams.[1] Their study highlights also several shortcomings, such as the lack of robust empirical evaluation and adequate tool support.

We should point out that CIT is a generic testing approach not only applicable in the context of SPL testing. In this general sense, CIT consists of four phases [21]: (i) *modeling* whose goal is to model the *System Under Test* (*SUT*) and its input space, (ii) *sampling* which produces a set of configurations that will be used for testing, (iii) *testing* that actually carries of the test based on different CIT parameters, and (iv) *analysis* where the results obtained are examined to identify faults and their underlying causes. In other words, the first two stages deal with the *what* should be tested, whereas the last two stages deal with the *how* should be tested [21].

Within the area of Search-Based Software Engineering a major research focus has been software testing [8, 22]. A recent overview by McMinn [10] highlights the major achievements made in the area and some of the open questions and challenges. We have performed a systematic mapping study whose focus is on SBSE

[1]http://www.uml.org/.

techniques applied to SPLs [23],[2] among such techniques are those based on evolutionary computation. Overall we found that almost all the research on evolutionary computation applied to SPL testing falls within the first two stages of CIT, modeling (based on feature models) and sampling (using different techniques), as we elaborate more on next section.

4 Combinatorial Interaction Testing for Software Product Lines

When Combinational Interaction Testing is applied to SPLs, the goal is to select a representative subset of products where interaction errors are more likely to occur rather than testing the complete product family [19]. In this section, we provide the basic terminology of CIT for SPLs,[3] use a simple evolutionary algorithm to illustrate CIT for the case of pairwise testing, and presents an overview of state of the art in CIT for SPL testing. In Sect. 5, we address the case when optimization of multiple objectives is considered.

4.1 Basic Terminology

Definition 4 (*t-set*) A t-set ts is a 2-tuple $[\texttt{sel}, \overline{\texttt{sel}}]$ representing a partially configured product, defining the selection of $\texttt{t}$ features of the feature list *FL*, i.e. $\texttt{ts.sel} \cup \texttt{ts.}\overline{\texttt{sel}} \subseteq FL \wedge \texttt{ts.sel} \cap \texttt{ts.}\overline{\texttt{sel}} = \emptyset \wedge |\texttt{ts.sel} \cup \texttt{ts.}\overline{\texttt{sel}}| = \texttt{t}$. We say t-set $\texttt{ts}$ is covered by feature set $\texttt{fs}$ iff $\texttt{ts.sel} \subseteq \texttt{fs.sel} \wedge \texttt{ts.}\overline{\texttt{sel}} \subseteq \texttt{fs.}\overline{\texttt{sel}}$.

Definition 5 (*Valid t-set*) A t-set $\texttt{ts}$ is valid in a feature model fm if there exists a valid feature set $\texttt{fs}$ that covers $\texttt{ts}$. The set of all valid t-sets for a feature model is denoted with $\mathcal{VTS}^{fm}$.

Definition 6 (*t-wise covering array*) A t-wise covering array $\texttt{tCA}$ for a feature model fm is a set of valid feature sets that covers all valid t-sets in $\mathcal{VTS}^{fm}$. Formally, $\texttt{tCA} \subseteq \mathcal{P}(\texttt{FS}^{fm})$ where $\forall \texttt{ts} \in \texttt{VTS}^{fm}, \exists \texttt{fs} \in \texttt{tCA}$ such that $\texttt{fs}$ covers $\texttt{ts}$.

Let us illustrate these concepts for *pairwise testing*, meaning when $\texttt{t = 2}$. From the feature model in Fig. 1, a valid 2-set is [{Driver},{Prim}]. It is valid because the selection of feature Driver and the non-selection of feature Prim do not violate any constraints. As another example, the 2-set [{Kruskal,DFS}, $\emptyset$] is

[2]An early version is available in [24].

[3]Definitions based on [12, 25].

Fig. 2 Graph Product Line 2-wise covering array example [26]

valid because there is at least one feature set, for instance fs1 in Table 1, where both features are selected. The 2-set [∅, {SCC,CC}] is also valid because there are valid feature sets that do not have any of these features selected, for instance feature sets fs0, fs2, and fs3. Notice, however, that the 2-set [∅, {Directed, Undirected}] is not valid. This is because feature GraphType is present in all the feature sets (mandatory child of the root) so either Directed or Undirected must be selected. In total, our running example has 418 valid 2-sets, so a 2-wise covering array must have all these pairs covered by at least one feature set. A covering array can be visually depicted as shown in Fig. 2 [26].

4.2 SPL Genetic Solver (SPLGS)

The *SPL Genetic Solver* (*SPLGS*) is a constructive genetic algorithm that computes pairwise covering arrays for SPLs based on a feature model that receives as input. It is based on the *Prioritized Genetic Solver* (*PGS*) by Ferrer et al. that takes into account priorities during the generation of test suites [27]. SPLGS extends and adapts PGS for generating test suites of product lines. In each iteration SPLGS adds a new feature set that contributes the most coverage to the partial solution until all

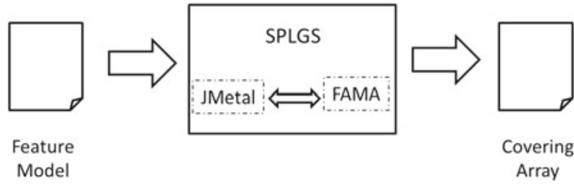

Fig. 3 Architecture of SPLGS

pairwise combinations are covered. SPLGS has been implemented using two framework tools: (i) jMetal [28], a Java framework aimed at the development, experimentation, and study of metaheuristics for solving optimization problems; and (ii) FAMA, an extensible framework for the representation and analysis of feature models [29]. The architecture of the SPLGS is presented in Fig. 3.

Algorithm 2 sketches the pseudocode of SPLGS. It takes as inputs the feature model FM. At the beginning, the test suite (`TS`) is initialized with an empty list (Line 4), and the set of remaining pairs (`RP`) is initialized with all valid pairs that need to be covered (Line 5). In each iteration of the external loop (Lines 6–24), the algorithm creates a random initial population of individuals (feature sets in our case) in (Line 8), and enters an inner loop which applies the traditional steps of a generational evolutionary algorithm (Lines 9–21). That is, some individuals are selected from the population $P(t)$, recombined, mutated, evaluated, and finally inserted in offspring population Q. If a generated offspring individual is not a valid feature set (i.e. it violates any constraint derived from the feature model), it is transformed into a valid one by applying a `Fix` operation (Line 15) provided by the FAMA tool [29]. The fitness value of an offspring individual is the number of pairs that remains to be covered, and hence it should be minimized (Line 16). In Line 19, the best individuals of $P(t)$ and Q are kept for the next generation $P(t+1)$. The internal loop is executed until a maximum number of evaluations is reached. Then, the best individual found is included in the test suite (Line 22) and `RP` is updated by removing the new pairs covered by the selected best solution (Line 23). Then, the external loop starts again until there is no pair left in the RP set. Finally, in Line 25 the computed test suite is returned. SPLGS has been shown to generate competitive test suites when compared against other leading CIT approaches for SPL, for further details refer to [30].[4]

[4]In [30] the algorithm is named PGS. We changed its name for this chapter to avoid confusions with the original algorithm PGS in [27] that was not designed for SPLs.

Algorithm 2. Pseudocode of SPLGS

```
1: proc SPLGS
2: Input: FM        // Input feature model
3: Output: TS       // Output test suite
4: TS ← ∅     // Empty test suite
5: RP ← pairs_to_cover(FM)   // Initialize the pairwise configurations
6: while not empty(RP) do
7:    t=0
8:    P(t) ← Create_Population() // P = population
9:    while  evals < totalEvals do
10:       Q ← ∅     // Q = auxiliary population
11:       for i ← 1 to (PGS.popSize / 2) do
12:          parents←Selection(P(t))
13:          offspring←Recombination(PGS.Pc,parents)
14:          offspring←Mutation(PGS.Pm,offspring)
15:          Fix(offspring)
16:          Evaluate_Fitness(offspring)
17:          Insert(offspring,Q)
18:       end for
19:       P(t+1) := Replace (Q,P(t))
20:       t= t + 1
21:    end while   //internal loop
22:    TS ← TS ∪ best_solution(P(t))
23:    RemovePairs(RP, best_solution(P(t)))
24: end while   //external loop
25: return TS
26: end_proc
```

4.3 State of the Art CIT for SPL Testing

There exists an important body of literature on CIT for SPL testing; however, only few examples rely on evolutionary algorithms. In this section we first present these approaches, followed by those that do not rely on evolutionary algorithms. In Sect. 5.4 we present the related work for Multi-Objective Evolutionary Algorithms for SPL testing.

Evolutionary Approaches. Ensan et al. [31] propose a genetic algorithm approach for test case generation for SPLs that uses a variation of cyclomatic complexity metric adapted to feature models and hence their goal is not to provide n-wise coverage. Henard et al. [32] propose an approach based on a (1 + 1) evolutionary algorithm that uses similarity heuristic as a viable alternative for t-wise coverage for coping with large scale feature models and large values of t up to 6. This approach is supported by a tool called PLEDGE that in addition provides a product line editor [33]. Work by Xu et al. [34] uses a genetic algorithm for continuous test augmentation. Their CONTESA tool incrementally generates test cases for branches that have not yet been covered by existing tests. Recent work by Henard et al. [35] creates so called *mutants* of a feature model which in addition to the original feature model are passed to a (1 + 1) evolutionary algorithm to produce test suites.

Non-Evolutionary Approaches. Garvin et al. [36] applied simulated annealing to combinatorial interaction testing for computing n-wise coverage for SPLs. Their algorithm CASA, performs three nested search strategies aiming at iteratively reducing the sizes of the test suites. Perrouin et al. propose an approach that first transforms t-wise coverage problems into Alloy programs and then uses Alloy's automatic instance generation to obtain covering arrays [37]. Oster et al. [38] propose MoSo-PoLiTe, an approach that transforms feature models into Constraint Solver Problems (CSP) to compute pairwise covering arrays. MoSo-PoLiTe can also include pre-selected products as part of the covering arrays. Hervieu et al. [39] follow a similar approach of using constraint programming for computing pairwise coverage. Regarding model based testing, the work by Lochau et al. [40] relates feature models with a reusable test model expressed with state charts to define and analyse feature dependencies and interactions. Cichos et al. [41] proposed an application of the so-called 150 % model, a model with all variable options included, whose goal is to provide complete test coverage for a given coverage criterion. Johansen et al. [25] propose a greedy approach to generate n-wise test suites that adapts Chvátal's algorithm to solve the set cover problem that makes several enhancements, for instance they parallelize the data independent processing steps. Calvagna et al. have developed CITLab [42], a tool for integrating multiple CIT approaches for SPLs.

5 Multi-objective SPL Testing

The approaches presented in Sect. 4 primarily focus on obtaining test suites that achieve complete coverage of the desired t strength. In other words, their single optimization objective is maximizing t-wise coverage. Though useful in many contexts, this single-objective perspective does not reflect the prevailing scenario where software engineers do face trade-offs among multiple and often conflicting objectives that represent technical and economical constraints. In this section, we first present a formalization of SPLs testing as a multi-objective optimization problem and provide a brief motivation example. Then as an example, we describe our exact algorithm to compute the optimal solutions for pairwise testing for coverage and test suite size optimization. And conclude with an overview of related multi-objective SPL testing approaches.

5.1 Multi-objective Optimization Formalization

There exists a wealth of literature in the context of Evolutionary Multi-Objective Optimization [43] and the application of *Search-Based Software Engineering* (*SBSE*) to software testing [44]. In this section we provide the formalization of SPL testing as a multi-objective optimization problem. Our definitions are based on

[45–47] and are generalizations of our previous work for the case of bi-objective pairwise testing [48].

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 subsets 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, e.g. $f^{fm} : \mathcal{DS}^{fm} \rightarrow \mathbb{N}$.

As examples, let us consider two objective functions:

- Coverage function. We want to maximize the number of t-wise sets covered by a test suite as follows:

$$f_1^{fm} : \mathcal{DS}^{fm} \rightarrow \mathbb{N},$$

$$f_1^{fm}(x) = |covers(x)|,$$

  where `covers` computes the t-wise sets covered by the feature sets of test suite `x`.
- Test suite size function. 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[5]:

$$F^{fm} : \mathcal{DS}^{fm} \rightarrow \mathcal{OS}^{fm}$$

$$F^{fm}(x) = \left(f_1^{fm}(x), f_2^{fm}(x), \ldots, f_n^{fm}(x)\right)$$

where $\mathcal{OS}$ is the corresponding `objective space`, in our context is $\mathcal{OS}^{fm} = \mathbb{N}^n$.

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: $u = F^{fm}(x) = (f_1^{fm}(x), f_2^{fm}(x), \ldots, f_n^{fm}(x))$.

---

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

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 [45].

Definition 11 (*Pareto dominance*) Let $x, y \in \mathcal{DS}^{fm}$, $u = F^{fm}(x) = (u_1, u_2, \ldots, u_n)$, and $v = F^{fm}(y) = (v_1, v_2, \ldots, v_n)$ for a feature model `fm`. Let $u \preccurlyeq v$ mean 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. Then we say that objective vector u `Pareto-dominates` objective vector `v` iff $u \preccurlyeq v$ and $v \npreccurlyeq u$.

Definition 12 (*Multi-Objective SPL n-wise testing problem*) A multi-objective n-wise 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 $\mathrm{x} \in \mathcal{DS}^{fm}$ is Pareto optimal iff it does not exist another $\mathrm{y} \in \mathcal{DS}^{fm}$ such that Pareto-dominates it, that is $F(y)^{fm} \preccurlyeq F(x)^{fm}$.

Definition 14 (*Pareto optimal set*) The Pareto optimal set P_*^{fm} of a multiobjective n-wise SPL testing problem for feature model *fm* and its vector function F^{fm} is: $P_*^{fm} = \{x \in DS^{fm} | \nexists x' \in DS^{fm} \text{such that}\, F^{fm}(x') \preccurlyeq F^{fm}(x)\}$.

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

5.2 An Example Scenario

Let us now motivate the importance of multi-objective optimization for SPL with a simple and illustrative example. Consider for instance our two objective functions f_1^{fm} and f_2^{fm}, as described above, that respectively represent the t-wise coverage and test suite size. On one hand we want to maximize t-wise coverage while *at the same time* we want to minimize the test suite size. Figure 4 shows the Pareto front for our running example GPL for the case of pairwise testing. Objective function f_1^{fm} is shown on the vertical axis as percentage of coverage pairs, while objective function f_2^{fm} is shown on the horizontal axis.

Taking a multi-objective approach and computing a Pareto front allows software engineers to select not just one solution, as in the case of single-objective techniques, but instead to select from an array the solution that best matches the economical and technological constraints of their testing context. In our example, some

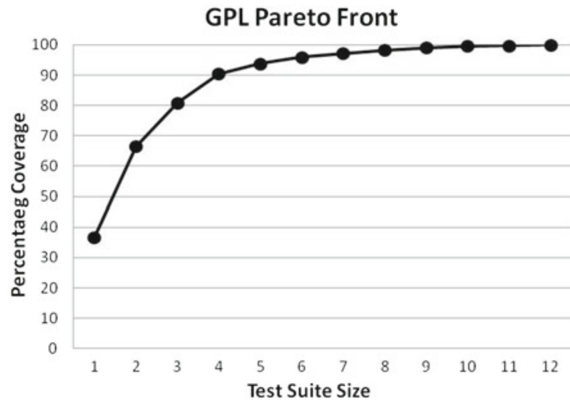

Fig. 4 Graph Product Line Paiwise Pareto Front

of the questions that can be answered and hence can help software engineers make informed decisions are:

– *What is the minimum size of a test suite that guarantees full pairwise coverage?* Clearly, from Fig. 4, this can only be achieved with 12 feature sets.
– *How many feature sets are needed to get a certain percentage coverage, for example 90 % coverage?* Again, from the information provided by the Pareto front we can affirm that only 4 products are needed to attain 90 % coverage.
– *If only 3 feature sets can be tested because of economical constraints, what is the maximum coverage that can be achieved?* Once more, using the information of the Pareto front, the maximum coverage is 80.86 %.

For this kind of concerns, software engineers not only get a single value, like the number of feature sets to test, but in addition they can also obtain a list of test suites that meet the desired criteria. In sharp contrast with single-objective approaches that can only provide a single solution. For example, the test suite shown in Fig. 2 is just but a single point that is mapped to the GPL Pareto front, in our case the rightmost point in Fig. 4 in the best scenario. Hence, for instance, the questions posed above cannot be addressed with a single-objective method. Next section we present an approach to compute the exact Pareto front which we used for computing the front for GPL.

5.3 Computation of Exact Pareto Fronts

In this section we present an overview of our work on computing exact Pareto fronts for SPL pairwise testing for two objectives, maximizing the pairwise coverage while minimizing the test suite size. For further details please refer to [49]. The algorithm we proposed for obtaining the optimal Pareto set is given in Algorithm 3, and is based on the work of Arito et al. [50] for solving a

multi-objective test suite minimization problem in regression testing. From the definitions in the previous subsection, recall that a Pareto optimal set is a set of non-dominated solutions each of which is not dominated by any other solution in the decision space, while the Pareto front is the projection of this set in the objective space, that is, a plot containing the values of the objective functions for each solution.

Algorithm 3 takes as input a feature model (Line 2) and computes the optimal Pareto set (Line 3). It first initializes the optimal set to empty (Line 4). Then it adds to the set two solutions that are always in the set: the empty solution with zero coverage (Line 5) and one arbitrary solution (Line 7) with coverage C_2^f, that is, the number 2-combinations of the set of features (Line 6). The algorithm then enters a loop (Lines 9–15) in which successive zero-one linear programs are generated (Line 12) for an increasing number of products starting at 2. A zero-one program is an integer program in which the variables can take as value either 0 or 1 [51]. In our case, this program serves to compute a solution which has the maximum coverage that can be obtained with i feature sets. In short, we describe this process in more detail.

Each mathematical model is then solved using an extended SAT solver (Line 13), in our case `MiniSat+`.[6] This solver provides a test suite with the maximum coverage for the given number of feature sets. This solution is subsequently added to the optimal Pareto set (Line 10), and the corresponding coverage is adjusted (Line 14). The algorithm stops when adding a new product to the test suite does not increase the coverage. The obtained Pareto optimal set is finally returned (Line 16).

Now we describe how to build the zero-one program for pairwise coverage. Let n be the fixed number of feature sets we want to compute and let f be the number of features of the input feature model FM. We use the set of decision variables $x_{i,j} \in \{0, 1\}$ where $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, f\}$, such that variable $x_{i,j}$ is 1 if feature set i has feature j selected and 0 otherwise. The zero-one program consists of four parts as described next.

1. **Constraints from feature model**. Recall that not all the combinations of features form valid products. The validity of any feature set denoted by a feature model FM can be expressed as a Boolean formula following the standard mapping to Conjunctive Normal Form (CNF) [12]. Each of the CNF clauses is then converted to a constraint of a zero-one program. First, let us define the Boolean vectors v and u as follows [52]:

[6]Available at URL: http://minisat.se/MiniSat+.html.

Algorithm 3. Algorithm for obtaining the Pareto optimal set

```
1: proc Pareto Optimal Set
2: Input: FM                    // Input feature model
3: Output: optimal_set          // Output Pareto optimal set
4: optimal_set ← {∅};
5: cov[0] ← 0;
6: cov[1] ← C_2^f;
7: sol ←arbitraryValidSolution(fm);
8: i ← 1;
9: while cov[i] ≠ cov[i − 1] do
10:    optimal_set ← optimal_set ∪ {sol};
11:    i ← i + 1;
12:    m ←prepareMathModel(fm,i);
13:    sol ←solveMathModel(m);
14:    cov[i] ← |covers(sol)|;
15: end while
16: return optimal_set
```

$$v_j = \begin{cases} 1 & \text{if feature } j \text{ appears in the clause}, \\ 0 & \text{otherwise}, \end{cases}$$
$$u_j = \begin{cases} 1 & \text{if feature } j \text{ appears negated in the clause}, \\ 0 & \text{otherwise}. \end{cases}$$

With the definitions of u and v, Eq. 1 describes how to write the constraint that corresponds to a CNF clause for the ith product.

$$\sum_{j=1}^{f} v_j(u_j(1 - x_{i,j}) + (1 - u_j)x_{i,j}) \geqslant 1 \quad (1)$$

As an illustration, let us suppose in our GPL running example that feature *Search* is the 8th feature in the feature list and *Num* is the 12th feature. The cross-tree constraint "*Num* requires *Search*", shown in Fig. 1, can be written in CNF with the clause ¬*Num* ∨ *Search* and its translation to a zero-one constraint is: $1 - x_{i,12} + x_{i,8} \geqslant 1$.

2. **Constraints for pairwise coverage per feature set**. Because our focus is pairwise coverage, we need to consider four possible combinations between two features: (i) both features unselected, (ii) first feature unselected and second feature selected, (iii) first feature selected and second feature unselected, (iv) both features selected.
   We introduce one variable in our program for each feature set, each pair of features and each of these four possibilities. The variables, called $c_{i,j,k,l}$, take value 1 if feature set i covers the pair of features j and k with the combination l.

The combination l is a number between 0 and 3 representing the selection configuration of the features according to the next mapping: $l = 0$, both unselected; $l = 1$, first unselected and second selected; $l = 2$, first selected and second unselected; and $l = 3$ both selected. The values of the variables $c_{i,j,k,l}$ depend on the values of $x_{i,j}$. In order to reflect this dependence in the mathematical program we add the following constraints for all $i \in \{1, \ldots, n\}$ and all $1 \leqslant j < k \leqslant f$:

$$2c_{i,j,k,0} \leqslant (1 - x_{i,j}) + (1 - x_{i,k}) \leqslant 1 + c_{i,j,k,0} \quad (2)$$

$$2c_{i,j,k,1} \leqslant (1 - x_{i,j}) + x_{i,k} \leqslant 1 + c_{i,j,k,1} \quad (3)$$

$$2c_{i,j,k,2} \leqslant x_{i,j} + (1 - x_{i,k}) \leqslant 1 + c_{i,j,k,2} \quad (4)$$

$$2c_{i,j,k,3} \leqslant x_{i,j} + x_{i,k} \leqslant 1 + c_{i,j,k,3} \quad (5)$$

3. **Constraints for pairwise coverage of all feature sets.** Variables $c_{i,j,k,l}$ inform about the coverage in one feature set. We need new variables to count the pairs covered when all the feature sets are considered. These variables are called $d_{j,k,l}$, and take value 1 when the pair of features j and k with combination l is covered by some product and 0 otherwise. This dependence between the $c_{i,j,k,l}$ variables and the $d_{j,k,l}$ variables is represented by the following set of inequalities for all $1 \leqslant j < k \leqslant f$ and $0 \leqslant l \leqslant 3$:

$$d_{j,k,l} \leqslant \sum_{i=1}^{n} c_{i,j,k,l} \leqslant n \cdot d_{j,k,l} \quad (6)$$

4. **Maximization goal.** Finally, the goal of our program is to maximize the pairwise coverage, which is given by the number of variables $d_{j,k,l}$ that are 1. This is expressed
as:

$$\max \sum_{j=1}^{f-1} \sum_{k=j+1}^{f} \sum_{l=0}^{3} d_{j,k,l} \quad (7)$$

In summary, the mathematical program is composed of the goal (7) subject to the $4(n+1)f(f-1)$ constraints given by (2) to (6) plus the constraints of the FM expressed with the inequalities (1) for each product. The number of variables of the program is $nf + 2(n+1)f(f-1)$. The solution to this zero-one linear program is a test suite with the maximum coverage that can be obtained with n feature sets.

Evaluation. We have evaluated our approach using a benchmark of 118 feature models publicly available in two open repositories [49], whose results are shown in Fig. 5. These feature models have number of feature sets that ranges from 16 to 640. We found that execution time does not grow linearly with the number of feature sets of the

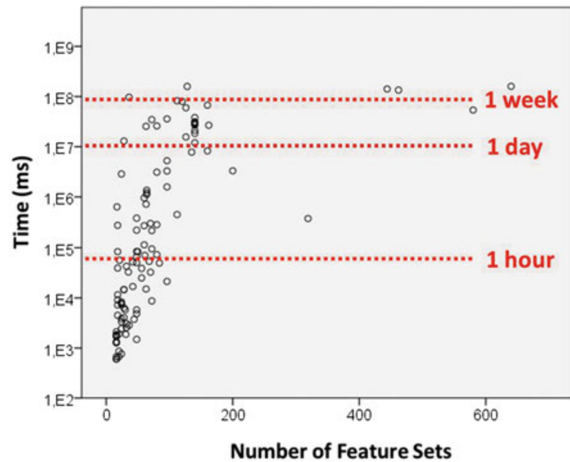

Fig. 5 Time (log scale) to compute Pareto optimal set versus number of feature sets

feature models, but instead it grows faster. Consequently we found scalability issues with our approach. Even though the majority of our examples finished within an hour, there were a significant portion that required a day and a few less that required a week of devoted computation in a standard desktop environment. Scalability issues of exact methods are the main reason for using approximate methods based on multi-objective evolutionary algorithms that we summarize in the next section.

5.4 Sate of the Art in Evolutionary Multi-objective Optimization for SPL Testing

We have performed a systematic mapping study on SBSE techniques applied to SPLs [23].[7] In this section we shortly summarize all the works found by this study and in addition describe the salient related work that uses evolutionary multi-objective algorithms for SPLs but not for testing.

Our previous work makes a comparison of four classical multi-objective evolutionary algorithms for SPL pairwise testing, namely: NSGA-II, PAES, MOCell, and SPEA2 [48]. In addition, this work analyzes the performance impact of three different seeding strategies that exploit different levels of domain knowledge to create the initial populations. We evaluated this work using 19 representative feature models from different application domains, ranging in number of features from 9 to 117, and in number of feature sets from 32 to 1,741,824. We found that the algorithms NSGA-II, SPEA2 or MOCell perform comparatively equal and perform best when using the seeding strategy that exploits the most domain knowledge (i.e. seeds the initial population based on a test suite computed using a single-objective algorithm).

[7] An early version is available in [24].

The work by Wang et al. present an approach to minimize test suites using weights in the fitness function [53], that is, it uses a *scalarizing function* that transforms a multi-objective problem to a single-objective one [47]. Their work uses three objectives: test minimization percentage, pairwise coverage, and fault detection capability. A similar approach is taken in recent work by Henard et al. that present an ad-hoc multi-objective algorithm whose fitness function is also scalarized [54]. Their work focuses also on maximizing coverage, minimizing test suite size, and minimizing cost. *We should remark that neither of these two approaches are multi-objective evolutionary algorithms in the strict sense*. Clearly, this is because these approaches compute only one single solution, that is, just a single point in the Pareto front. Incidentally, we should point out there is an extensive body of work on the downsides of scalarization in multi-objective optimization (e.g. [55]). 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.

We should also point out that there is a considerable number of applications of multi-objective evolutionary algorithms but outside of the testing activities of SPL development. A common task where these algorithms is employed is in product configuration. For example, Cruz et al. employ the multi-objective algorithm NSGA-II to create and manage product portfolios based on customer satisfaction and costs [56]. As another example, the work by Sayyad et al. performs a more thorough exhaustive application and analysis of multi-objective evolutionary algorithms for configuration tasks [57, 58]. Our own previous work has also explored using several classical multi-objective evolutionary algorithms for the configuration of dynamic product lines for mobile applications [59].

For sake of completeness, we should also indicate ongoing work on exact multi-objective method by Olaechea et al. who propose an exact method to compute Pareto fronts showing their capability to handle small and medium size problems and provide basic guidelines for choosing either exact or evolutionary approaches [60]. Similarly, Murashkin et al. present a tool for the visualization and exploration of variants in a multi-dimensional space but do not address SPL testing issues [61].

6 Evolutionary Testing of SPLs in Practice

The most common scenario for the development of SPLs in industrial setting comes after the realization that developing and maintaining multiple similar systems, an approach called "*Clone and Own*", is not economically feasible [62]. The main goals of reverse engineering a SPL from a set of similar software systems are: (i) capture the knowledge of what is common and what is variable (e.g. commonality and variability) in all the artifacts employed throughout the development life-cycle, and (ii) express with a feature model all the valid feature combinations required for the SPL. The result of the reverse engineering process is illustrated in Fig. 6.

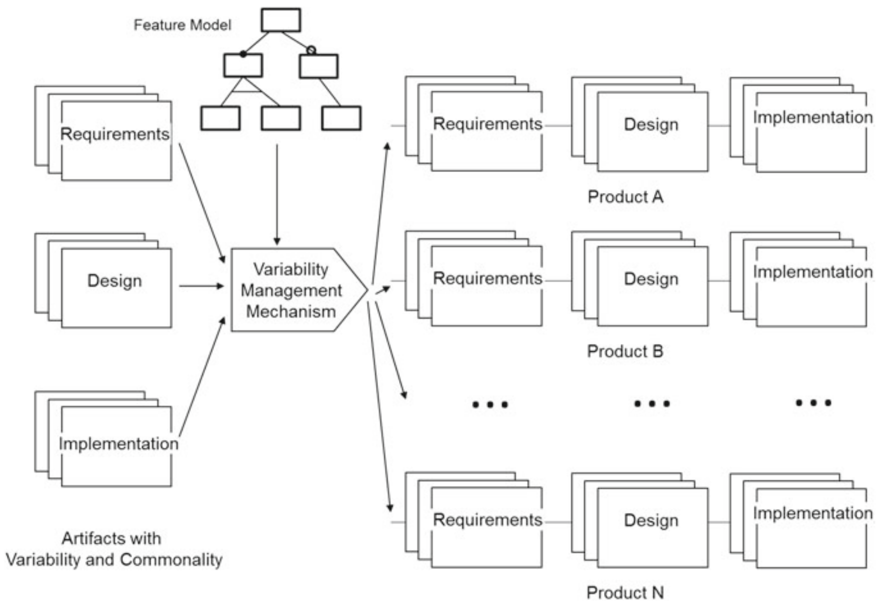

Fig. 6 Overview of reversed engineered SPL

There has been extensive work over the last two decades on how to capture the variability and commonality knowledge for SPLs (i.e. Variability Management Mechanism in Fig. 6), for a summary see for example [63]. Regarding the extraction of feature models, recent work by Lopez-Herrejon et al. also uses evolutionary algorithms in combination of information retrieval metrics for obtaining feature models based on the feature combinations [64, 65]. Alternative approaches can obtain feature models based on generic propositional logic constraints [66].

Once a SPL infrastructure has been put in place, SPL testing can proceed. There has been recent accounts of SPL testing in industrial settings, some of them relying on evolutionary approaches. For instance, Wang et al. report on an experience in the application of multi-objective algorithms for a video conference application [67]. The common trend in such experiences is that the critical factor is eliciting the right feature models from the software engineers, this is so because the information of the valid feature combinations are commonly not well documented if at all. The application of the testing techniques in general do not require expensive hardware or software infrastructure, as in the common cases their execution takes a few minutes or hours in typical off-the-shelf desktop computers. There are, however, empirical and theoretical studies on large scale feature models, mostly of academic interest, that consider large number of features, products, or higher array strengths (e.g. up to 6) which show approaches for coping with scalability issues (e.g. [32]).

7 Open Challenges and Questions

In this section we describe what we consider the most salient open challenges and questions for SPL testing with evolutionary techniques.

Multi-objective optimization. There is an extensive body of research literature in multi-objective optimization that remains largely untapped, for an overview see for example [45, 46]. An open question is whether other multi-objective evolutionary algorithms can yield better results and under which circumstances. In addition, further studies are needed that explore dealing with more optimization objectives which can for instance include information such as control-flow or non-functional properties. Furthermore, it is an open question if so-called *many-objective* optimization algorithms, those that deal with four or more objectives, can also be effectively applied to the context of SPL testing. Of crucial importance is their scalability as the complexity of the feature models or the strength of covering arrays increases.

Need of community-wide testing benchmarks and comparison frameworks. We found that the majority of works employs feature models extracted from common repositories such as SPLOT.[8] However, the selection of which feature models to analyze in each paper appears to be arbitrary most of the times. The first steps towards a benchmark for CIT SPL testing are advocated in [30]. Work by Perrouin et al. proposed a comparison framework which is applied to two different approaches [68]. However, comparisons can only be made per feature model, which makes it infeasible to identify which approach performs overall better or under what characteristics of the feature models [30]. Without a proper and fair benchmark and comparison framework, the progress in the research and its transfer to industry are severely hampered.

Exploiting more SPL domain knowledge. Because of the typically large number of individual systems of a SPL, any information that could be exploited to reduce the search effort is worth of consideration. For example, Haslinger et al. leverage information from feature models to speed up the computation of covering arrays by eliminating redundant *t*-sets [69, 70]. As other examples, the work by Xu et al. that exploits static analysis techniques for achieving coverage more effectively [34], and the work by Lopez-Herrejon et al. that studies seeding strategies [48]. It remains an open question, whether any of the analysis techniques recently surveyed by Thüm et al. (see [71]) could be exploited for this purpose. Also, recent work by Fischer et al. (see [72]) computes traceability links from features and feature interactions to the artifacts that realize them. It is an open question whether such traceability information could also be helpful to prune the search space.

Test suite prioritization. Johansen et al. [73] propose a greedy algorithm that adds weights to products to guide the computation of the t-wise sets. These weights are meant to represent priority values such as commercial importance. An alternative parallel evolutionary algorithm was proposed by Lopez-Herrejon et al. for

[8] http://www.splot-research.org/.

this scheme that can produce smaller test suites [74]. Additionally, once test suites have been computed their feature sets can be ordered according to some criteria. For example, Al-Hajjaji et al. propose a prioritization based on similarity [75], while Sánchez et al. compare five SPL-specific prioritization criteria and analyze their effect in detecting faults in order to provide faster feedback and reduce debugging efforts [76]. Another example is the recent work by Wang et al. who proposed a scalarized four-objective function to prioritize quasi-pairwise (not considering all four possible combinations of a 2-wise set) tests suites [67]. Test suite prioritization has a long research literature for single software systems (e.g. [77]) which has not been thoroughly researched within the context of SPLs. Some open research issues are prioritization using combinations of clustering techniques and classical multi-objective evolutionary algorithms (instead of scalarization approaches) that exploit the values obtained for instance from non-functional properties.

Supporting testing and analysis phases of CIT. As mentioned before, CIT consists of four phases [21]. However, the current focus for SPL has mostly been on the first two: modeling relying on feature models, and sampling as summarized in Sects. 4.3 and 5.4. Hence, there is a dire need of research and practice that addresses this limitation. Tools such as EvoSuite[9] could be leveraged as starting point for such tasks.

8 Conclusions

Software Product Lines are an emerging software development paradigm that aims to provide a systematic and methodological reuse of all the assets involved in the development of families of software systems, where products share common functionality but also can have unique distinct features. The proven benefits of SPL practices (e.g. [3]) have resulted in an increasing interest both by researchers and practitioners on effective techniques and tools for adequately testing SPLs. The most important challenge faced is dealing with the generally large number of products, i.e. combinations of features, which makes an infeasible alternative testing individually each one of them.

Recent surveys not only highlight the increasing interest in the area but also several shortcomings and opportunities that exist on the field [4, 5, 7, 16–18]. Within the area of Search-Based Software Engineering a major research focus has been software testing [8, 10, 22], also including evolutionary computation techniques. However, most of the applications are for one-off systems rather than SPLs. The goal of this chapter is to provide an overview of the state of the art in SPL testing and framing evolutionary approaches within that context. We have put forward several challenges and open questions that we believe could be fruitful avenues for further research and practice.

[9]http://www.evosuite.org/.

Acknowledgments This research is partially funded by the Austrian Science Fund (FWF) projects P 25513-N15, P 25289-N15, and Lise Meitner Fellowship M1421-N15, and by the Spanish Ministry of Economy and Competitiveness and FEDER under contract TIN2011-28194 and fellowship BES-2012-055967. It is also partially founded by projects 8.06/5.47.4142 (collaboration with the VSB-Tech. Univ. of Ostrava) and 8.06/5.47.4356 (Andalusian Agency of Public Works).

References

1. Pohl, K., Bockle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin (2005)
2. Batory, D.S., Sarvela, J.N., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Softw. Eng. **30**(6), 355–371 (2004)
3. van der Linden, F., Schmid, K., Rommes, E.: Software Product Lines in Action—The Best Industrial Practice in Product Line Engineering. Springer, Berlin (2007)
4. Engström, E., Runeson, P.: Software product line testing—A systematic mapping study. Inf. Softw. Technol. **53**(1), 2–13 (2011)
5. da Mota Silveira Neto, P.A., do Carmo Machado, I., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R.: A systematic mapping study of software product lines testing. Inf. Softw. Technol. **53**(5), 407–423 (2011)
6. Lee, J., Kang, S., Lee, D.: A survey on software product line testing. 16th International Software Product Line Conference, pp. 31–40 (2012)
7. do Carmo Machado, I., McGregor, J.D., de Almeida, E.S.: Strategies for testing products in software product lines. ACM SIGSOFT Softw. Eng. Notes **37**(6), 1–8 (2012)
8. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: trends, techniques and applications. ACM Comput. Surv. **45**(1), 11 (2012)
9. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Springer, Berlin (2003)
10. McMinn, P.: Search-based software testing: past, present and future. In: ICST Workshops, pp. 153–163. IEEE Computer Society (2011)
11. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
12. Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010)
13. Lopez-Herrejon, R.E., Batory, D.S.: A standard problem for evaluating product-line methodologies. In: Bosch, J. (ed.) GCSE. Volume 2186 of Lecture Notes in Computer Science, pp. 10–24. Springer, Berlin (2001)
14. Michalewicz, Z., Fogel, D.B.: How to Solve It: Modern Heuristics, 2nd edn. Springer, Berlin (2010)
15. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
16. da Mota Silveira Neto, P.A., Runeson, P., do Carmo Machado, I., de Almeida, E.S., de Lemos Meira, S.R., Engström, E.: Testing software product lines. IEEE Software **28**(5), 16–20 (2011)
17. Wohlin, C., Runeson, P., da Mota Silveira Neto, P.A., Engström, E., do Carmo Machado, I., de Almeida, E.S.: On the reliability of mapping studies in software engineering. J. Syst. Softw. **86**(10), 2594–2610 (2013)
18. do Carmo Machado, I., McGregor, J.D., Cavalcanti, Y.C., de Almeida, E.S.: On strategies for testing software product lines: a systematic literature review. Inf. Softw. Technol. **56**(10), 1183–1199 (2014)

19. Cohen, M.B., Dwyer, M.B., Shi, J.: Constructing interaction test suites for highly-configurable systems in the presence of constraints: a greedy approach. IEEE Trans. Softw. Eng. **34**(5), 633–650 (2008)
20. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. **43**(2), 11:1–11:29 (February 2011)
21. Yilmaz, C., Fouché, S., Cohen, M.B., Porter, A.A., Demiröz, G., Koc, U.: Moving forward with combinatorial interaction testing. IEEE Comput. **47**(2), 37–45 (2014)
22. de Freitas, F.G., de Souza, J.T.: Ten years of search based software engineering: a bibliometric analysis. In: Cohen, M.B., Cinnéide, M.Ó. (eds.) SSBSE. Volume 6956 of Lecture Notes in Computer Science, pp. 18–32. Springer, Berlin (2011)
23. Lopez-Herrejon, R.E., Linsbauer, L., Egyed, A.: A systematic mapping study of search-based software engineering for software product lines. Inf. Softw. Technol. J. (to appear)
24. Lopez-Herrejon, R.E., Ferrer, J., Chicano, F., Linsbauer, L., Egyed, A., Alba, E.: A hitchhiker's guide to search-based software engineering for software product lines. CoRR **abs/1406.2823** (2014)
25. Johansen, M.F., Haugen, Ø., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. 16th International Software Product Line Conference, pp. 46–55 (2012)
26. Lopez-Herrejon, R.E., Egyed, A.: Towards interactive visualization support for pairwise testing software product lines. In: Telea, A., Kerren, A., Marcus, A. (eds.) VISSOFT, pp. 1–4. IEEE (2013)
27. Ferrer, J., Kruse, P.M., Chicano, J.F., Alba, E.: Evolutionary algorithm for prioritized pairwise test data generation. InL: Soule, T., Moore, J.H. (eds.) GECCO, pp. 1213–1220. ACM (2012)
28. Durillo, J.J., Nebro, A.J.: jmetal: a java framework for multi-objective optimization. Adv. Eng. Softw. **42**(10), 760–771 (2011)
29. Trinidad, P., Benavides, D., Ruiz-Cortes, A., Segura, S., Jimenez, A.: Fama framework. In: Software Product Line Conference, 2008. SPLC'08. 12th International (Sept.), pp. 359–359
30. Lopez-Herrejon, R.E., Ferrer, J., Chicano, F., Haslinger, E.N., Egyed, A., Alba, E.: Towards a benchmark and a comparison framework for combinatorial interaction testing of software product lines. CoRR **abs/1401.5367** (2014)
31. Ensan, F., Bagheri, E., Gasevic, D.: Evolutionary search-based test generation for software product line feature models. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE. Volume 7328 of Lecture Notes in Computer Science, pp. 613–628. Springer, Berlin (2012)
32. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., Traon, Y.L.: Bypassing the combinatorial explosion: using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans. Softw. Eng. **40**(7), 650–670 (2014)
33. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Traon, Y.L.: Pledge: a product line editor and test generation tool. In: SPLC Workshops, pp. 126–129. ACM (2013)
34. Xu, Z., Cohen, M.B., Motycka, W., Rothermel, G.: Continuous test suite augmentation in software product lines. In: Proceedings SPLC, pp. 52–61 (2013)
35. Henard, C., Papadakis, M., Traon, Y.L.: Mutation-based generation of software product line test configurations. In: SSBSE, pp. 92–106 (2014)
36. Garvin, B.J., Cohen, M.B., Dwyer, M.B.: Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Softw. Eng. **16**(1), 61–102 (2011)
37. Perrouin, G., Sen, S., Klein, J., Baudry, B., Traon, Y.L.: Automated and scalable t-wise test case generation strategies for software product lines. In: ICST, pp. 459–468. IEEE Computer Society (2010)
38. Oster, S., Markert, F., Ritter, P.: Automated incremental pairwise testing of software product lines. In Bosch, J., Lee, J. (eds.) SPLC. Volume 6287 of Lecture Notes in Computer Science, pp. 196–210. Springer, Berlin (2010)

39. Hervieu, A., Baudry, B., Gotlieb, A.: Pacogen: automatic generation of pairwise test configurations from feature models. In: Dohi, T., Cukic, B. (eds.) ISSRE, pp. 120–129. IEEE (2011)
40. Lochau, M., Oster, S., Goltz, U., Schürr, A.: Model-based pairwise testing for feature interaction coverage in software product line engineering. Softw. Qual. J. **20**(3–4), 567–604 (2012)
41. Cichos, H., Oster, S., Lochau, M., Schürr, A.: Model-based coverage-driven test suite generation for software product lines. In: Whittle, J., Clark, T., Kühne, T. (eds.) MoDELS. Volume 6981 of Lecture Notes in Computer Science, pp. 425–439. Springer, Berlin (2011)
42. Calvagna, A., Gargantini, A., Vavassori, P.: Combinatorial testing for feature models using citlab. In: ICST Workshops, pp. 338–347 (2013)
43. Coello, C.C.: Evolutionary multi-objective optimization website. http://delta.cs.cinvestav.mx/ccoello/EMOO/
44. Zhang, Y.: Search Based Software Engineering Repository. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/
45. Coello, C.C., Lamont, G.B., Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-objective Problems, 2nd edn. Genetic and Evolutionary Computation. Springer, Berlin (2007)
46. Deb, K.: Multi-objective Optimization Using Evolutionary Algorithms, 1st edn. Wiley, New York (June 2001)
47. Zitzler, E.: Evolutionary multiobjective optimization. In: Handbook of Natural Computing, pp. 871–904 (2012)
48. Lopez-Herrejon, R.E., Ferrer, J., Chicano, F., Egyed, A., Alba, E.: Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, 6–11 July 2014, pp. 387–396. IEEE (2014)
49. Lopez-Herrejon, R.E., Chicano, J.F., Ferrer, J., Egyed, A., Alba, E.: Multi-objective optimal test suite computation for software product line pairwise testing. In: ICSM, pp. 404–407. IEEE (2013)
50. Arito, F., Chicano, F., Alba, E.: On the application of sat solvers to the test suite minimization problem. In: Proceedings of the Symposium of Search Based Software Engineering. Volume 7515 of LNCS, pp. 45–59 (2012)
51. Wolsey, L.A.: Integer Programming. Wiley, New York (1998)
52. Sutton, A.M., Whitley, L.D., Howe, A.E.: A polynomial time computation of the exact correlation structure of k-satisfiability landscapes. In: Proceedings of GECCO, pp. 365–372 (2009)
53. Wang, S., Ali, S., Gotlieb, A.: Minimizing test suites in software product lines using weight-based genetic algorithms. In: GECCO, pp. 1493–1500 (2013)
54. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Traon, Y.L.: Multi-objective test generation for software product lines. In: Proceedings of SPLC, pp. 62–71 (2013)
55. Marler, R., Arora, J.: Survey of multi-objective optimization methods for engineering. Struct. Multi. Optim. **26**(6), 369–395 (2004)
56. Cruz, J., Neto, P.S., Britto, R., Rabelo, R., Ayala, W., Soares, T., Mota, M.: Toward a hybrid approach to generate software product line portfolios. In: IEEE Congress on Evolutionary Computation, pp. 2229–2236 (2013)
57. Sayyad, A.S., Menzies, T., Ammar, H.: On the value of user preferences in search-based software engineering: a case study in software product lines. In: Proceedings of ICSE, pp. 492–501 (2013)
58. Sayyad, A.S., Ingram, J., Menzies, T., Ammar, H.: Scalable product line configuration: a straw to break the camel's back. In: ASE, pp. 465–474 (2013)
59. Pascual, G.G., Lopez-Herrejon, R.E., Pinto, M., Fuentes, L., Egyed, A.: Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. J. Syst. Softw. (2015, to appear)

60. Olaechea, R., Rayside, D., Guo, J., Czarnecki, K.: Comparison of exact and approximate multi-objective optimization for software product lines. In: Gnesi, S., Fantechi, A. (eds.) 18th International Software Product Line Conference, SPLC'14, pp. 92–101. Florence, Italy, 15–19 Sept 2014. ACM (2014)
61. Murashkin, A., Antkiewicz, M., Rayside, D., Czarnecki, K.: Visualization and exploration of optimal variants in product line engineering. In: Proceedings of SPLC, pp. 111–115 (2013)
62. Dubinsky, Y., Rubin, J., Berger, T., Duszynski, S., Becker, M., Czarnecki, K.: An exploratory study of cloning in industrial software product lines. In: Cleve, A., Ricca, F., Cerioli, M. (eds.) CSMR, pp. 25–34. IEEE Computer Society (2013)
63. Chen, L., Babar, M.A.: A systematic review of evaluation of variability management approaches in software product lines. Inf. Softw. Technol. **53**(4), 344–362 (2011)
64. Lopez-Herrejon, R.E., Linsbauer, L., Galindo, J.A., Parejo, J.A., Benavides, D., Segura, S., Egyed, A.: An assessment of search-based techniques for reverse engineering feature models. J. Syst. Softw. Spec. Issue Search-Based Softw. Eng. (2015)
65. Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A.: Feature model synthesis with genetic programming. In: Goues, C.L., Yoo, S. (eds.) Search-Based Software Engineering—6th International Symposium, SSBSE 2014, Fortaleza, Brazil, 26–29 Aug 2014. Proceedings. Volume 8636 of Lecture Notes in Computer Science, pp. 153–167. Springer, Berlin (2014)
66. She, S., Ryssel, U., Andersen, N., Wasowski, A., Czarnecki, K.: Efficient synthesis of feature models. Inf. Softw. Technol. **56**(9), 1122–1143 (2014)
67. Wang, S., Buchmann, D., Ali, S., Gotlieb, A., Pradhan, D., Liaaen, M.: Multi-objective test prioritization in software product line testing: an industrial case study. In: Gnesi, S., Fantechi, A. (eds.) 18th International Software Product Line Conference, SPLC'14, pp. 32–41. Florence, Italy, 15–19 Sept 2014. ACM (2014)
68. Perrouin, G., Oster, S., Sen, S., Klein, J., Baudry, B., Traon, Y.L.: Pairwise testing for software product lines: comparison of two approaches. Softw. Qual. J. **20**(3–4), 605–643 (2012)
69. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: Using feature model knowledge to speed up the generation of covering arrays. In: Gnesi, S., Collet, P., Schmid, K. (eds.) VaMoS, p. 16. ACM (2013)
70. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: Improving casa runtime performance by exploiting basic feature model analysis. CoRR **abs/1311.7313** (2013)
71. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. **47**(1), 6 (2014)
72. Fischer, S., Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A.: Enhancing clone-and-own with systematic reuse for developing software variants. 30th International Conference on Software Maintenance and Evolution (2014, to appear)
73. Johansen, M.F., Haugen, Ø., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. In: SPLC (1), pp. 46–55 (2012)
74. Lopez-Herrejon, R.E., Ferrer, J., Chicano, F., Haslinger, E.N., Egyed, A., Alba, E.: A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In: Arnold, D. V. (ed.) Genetic and Evolutionary Computation Conference, GECCO'14, Vancouver, BC, Canada, 12–16 July 2014, pp. 1255–1262. ACM (2014)
75. Al-Hajjaji, M., Thüm, T., Meinicke, J., Lochau, M., Saake, G.: Similarity-based prioritization in software product-line testing. In: Gnesi, S., Fantechi, A. (eds.) 18th International Software Product Line Conference, SPLC'14, pp. 197–206. Florence, Italy, 15–19 Sept 2014. ACM (2014)
76. Sánchez, A.B., Segura, S., Cortés, A.R.: A comparison of test case prioritization criteria for software product lines. In: ICST, pp. 41–50 (2014)
77. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Softw. Test., Verif. Reliab. **22**(2), 67–120 (2012)

Author Biographies

Dr. Roberto Erick Lopez-Herrejon is currently a senior postdoctoral researcher at the Johannes Kepler University in Linz, Austria. He was an Austrian Science Fund (FWF) Lise Meitner Fellow (2012–2014) at the same institution. From 2008 to 2014 he was an External Lecturer at the Software Engineering Masters Programme of the University of Oxford, England. From 2010 to 2012 he held an FP7 Intra-European Marie Curie Fellowship sponsored by the European Commission. He obtained his Ph.D. from the University of Texas at Austin in 2006, funded in part by a Fulbright Fellowship sponsored by the U.S. State Department. From 2005 to 2008, he was a Career Development Fellow at the Software Engineering Centre of the University of Oxford sponsored by Higher Education Founding Council of England (HEFCE). His expertise is software product lines, variability management, feature oriented software development, model driven software engineering, and consistency checking.

Mr. Javier Ferrer is a Ph.D. candidate. He had his 5-year Combined Bachelor's and Master's Engineering Degree by the University of Malaga. He has also obtained his M.Sc. in computer science (artificial intelligence) and his postgraduate certificate in education by the same university. His research interests are mainly related to metaheuristics optimization techniques. Specifically, he has several publications on the search based software engineering field. Overall, he has more than 20 publications including journal articles, conference papers, and book chapters. Currently his main research line is focused on the evolutionary testing domain.

Francisco Chicano is an associate professor in the Department of Languages and Computing Sciences of the University of Malaga, Spain. He studied Computer Science (2003) and Ph.D. in Computer Science (2007) at University of Malaga, and Physics (2014) in the National Distance Education University. His research interests include the application of randomized search techniques to Software Engineering problems. In particular, he contributed to the domains of software testing, model checking and software project scheduling. He also works on the landscapes theory of combinatorial optimization problems and the application of theoretical results to the design of new search algorithms and operators. He is the author of more than 70 refereed publications, has 3 best paper awards and has served on more than 30 program committees. He has served as Program Chair in the EvoCOP 2015 conference, as Track Chair in GECCO 2013 and GECCO 2015 and as Guest Editor in Special Issues of Evolutionary Computation (MIT), Journal of Systems and Software and Algorithmica. He is frequent reviewer in more than 10 international top journals and during the course 2014/2015 is Faculty Affiliate in the Colorado State University.

Alexander Egyed is a Full Professor at the Johannes Kepler University (JKU), Austria. He received his Doctorate degree from the University of Southern California, USA and previously work at Teknowledge Corporation, USA (2000–2007) and University College London, UK (2007–2008). Dr. Egyed's work has been published at over a hundred refereed scientific books, journals, conferences, and workshops, with over 3500 citations to date. He was recognized as a Top 1 % scholar in software engineering in the Communications of the ACM, Springer Scientometrics, and Microsoft Academic Search. He was also named an IBM Research Faculty Fellow in recognition to his contributions to consistency checking, received a Recognition of Service Award from the ACM, Best Paper Awards from COMPSAC and WICSA, and an Outstanding Achievement Award from the USC. He has given many invited talks including four keynotes, served on scientific panels and countless program committees, and has served as program (co-) chair, steering committee member, and editorial board member. He is a member of the IEEE, IEEE Computer Society, ACM, and ACM SigSoft.

Prof. Enrique Alba had his degree in engineering and Ph.D. in Computer Science in 1992 and 1999, respectively, by the University of Málaga (Spain). He works as a Full Professor in this university with different teaching duties: data communications, distributed programing, software quality, and also evolutionary algorithms, bases for R+D+i and smart cities at graduate and master programs. Prof. Alba leads an international team of researchers in the field of complex optimization/learning with applications in smart cities, bioinformatics, software engineering, telecoms, and others. In addition to the organization of international events (ACM GECCO, IEEE IPDPS-NIDISC, IEEE MSWiM, IEEE DS-RT, …) Prof. Alba has offered dozens postgraduate courses, multiple seminars in more than 20 international institutions, and has directed several research projects (6 with national funds, 5 in Europe, and numerous bilateral actions). Also, Prof. Alba has directed 7 projects for innovation and transference to the industry (OPTIMI, Tartessos, ACERINOX, ARELANCE, TUO, INDRA, ZED) and presently he also works as invited professor at INRIA, the Univ. of Luxembourg, and Univ. of Ostrava. He is editor in several international journals and book series of Springer-Verlag and Wiley, as well as he often reviews articles for more than 30 impact journals. He has published 80 articles in journals indexed by Thomson ISI, 17 articles in other journals, 40 papers in LNCS, and more than 250 refereed conferences. Besides that, Prof. Alba has published 11 books, 39 book chapters, and has merited 6 awards to his professional activities. Pr. Alba’s H index is 41, with more than 8000 cites to his work.