A First Systematic Mapping Study on Combinatorial Interaction Testing for Software Product Lines

Roberto E. Lopez-Herrejon*, Stefan Fischer*, Rudolf Ramler† and Alexander Egyed*
* Institute for Software Systems Engineering
Johannes Kepler University Linz, Austria
Email: {roberto.lopez, stefan.fischer, alexander.egyed}@jku.at
†Software Competence Center Hagenberg, Austria
Email: rudolf.ramler@scch.at

***Abstract*—Software Product Lines (SPLs) are families of related software systems distinguished by the set of features each one provides. Over the past decades SPLs have been the subject of extensive research and application both in academia and industry. SPLs practices have proven benefits such as better product customization and reduced time to market. Testing SPLs pose additional challenges stemming from the typically large number of product variants which make it infeasible to test every single one of them. In recent years, there has been an extensive research on applying Combinatorial Interaction Testing (CIT) for SPL testing. In this paper we present the first systematic mapping study on this subject. Our research questions aim to gather information regarding the techniques that have been applied, the nature of the case studies used for their evaluation, and what phases of CIT have been addressed. Our goal is to identify common trends, gaps, and opportunities for further research and application.**

I. Introduction

Software Product Lines (SPLs) are families of related systems whose members are distinguished by the set of features they provide [1], [2]. Over the last two decades, extensive research and practice both in academia and industry clearly attest to the significant benefits of applying SPL practices [2], [3], [4]. Among these benefits are better customization, improved software reuse, and faster time to market. *Variability* is the capacity of software artifacts to vary and its effective management and realization lie at the core of successful SPL development [5]. However, variability poses special challenges for SPL testing because variable artifacts can typically result in a large number of different software products which most of the times cannot be all tested individually.

Combinatorial Interaction Testing (CIT) is a testing method that models a *System Under Test (SUT)* as a set of factors (choice points or parameters) each taking its values from a particular domain [6]. CIT has a wealth of research literature and has been successfully applied to different types of software systems [7], [8], [9]. It is precisely this capability to deal with multiple combinations of parameters that has attracted the interest of the SPL community. In recent years, there has been an increasing number of publications that propose applications of CIT concepts and techniques for SPL testing. When applied to SPLs, the set of factors to consider for the combinations are the features that the systems of an SPL provide which are expressed using *variability models*.

The increasing number of publications of CIT for SPLs what prompted us to perform a *systematic mapping study* to provide an overview of the research on this subject [10], [11], [12]. In contrast with a *systematic literature review* whose goal is primarily to identify best practice [10], [12], [13], [14], our general goal was to identify the quantity and the type of research and results available, and consequently unveil open research problems and opportunities for the CIT and SPL communities. For this first mapping study our concrete goals were to identify the techniques, algorithms and tools that have been used for CIT of SPLs, the CIT phases where they have been used [6], the publication fora employed, and to catalogue the case studies used for evaluation and their provenance. The latter goal aimed at identifying common case studies that could be used as the basis for a benchmark for comparing and contrasting different approaches.

Our study corroborated the increasing interest in applying CIT techniques to SPLs. We found that most of the research has been focused on finding the products that represent the SPL in the testing scenario based on a variability model, rather than on actually performing the tests. Our study also found that the most common technique used is greedy algorithms. Furthermore we corroborated the need for more robust empirical evaluations including a community-wide benchmark to guide the comparison of different approaches and to identify topics to be considered in future studies. Part of our future work is adding new research questions, perform a more in depth analysis and a revised search for primary sources.

The paper is structured as follows. Section II presents the basic concepts of the most common variability models and our canonical running example. Section III makes the connection between CIT and SPL terminology. Section IV presents the process we followed in our systematic mapping study. It describes the research questions addressed, how the search was performed, the classification scheme used, and how the data was extracted and analysed. Section V presents the results obtained for each research question, while Section VI analyzes

2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
4th International Workshop on Combinatorial Testing (IWCT 2015)
978-1-4799-1885-0/15/$31.00

the findings and highlights open questions and avenues worth of further investigation. Section VII identifies possible threats to validity for our mapping study. Section VIII describes related literature reviews and surveys of CIT and SPL testing. Section IX summarizes the conclusions of our study and sketches our future work.

II. Feature Models and Running Example

There are several alternatives of variability models to express the set of feature combinations that constitute a SPL [15]. However, the most common one is *feature models* [16], which we use to illustrate and define the relevant concepts of CIT applied to SPLs. In this type of diagrams, features are depicted as labelled boxes and their relationships as lines, collectively forming a tree-like structure.

As a running example, we use a canonical SPL that has been extensively used in the SPL community. This SPL is called the *Graph Product Line (GPL)* and its products are combinations of basic graph algorithms and graph types [17]. Figure 1 shows the feature model of GPL. In this example, a product 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`), with optional weights (`Weight`). Two graph traversal algorithms (`Search`) can be optionally provided: Depth First Search (`DFS`) or Breadth First Search (`BFS`). A product 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 just one parent feature and can have any number of child features. A child feature can only be selected in a feature combination of a valid product if its parent feature 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.

A feature can be *mandatory* or *optional* which means that if its parent is selected respectively it must be selected or that it may or may not be selected. Mandatory features are denoted with a filled circle while optional feature are denoted with empty circles. For example, features `Algorithms` and `GraphType` are mandatory, while feature `Search` is optional.

Furthermore, features can be grouped together in two different ways. *Exclusive-or groups* 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 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 groups* 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. An example is feature `Algorithms` from which at least one of the features `Num`, `CC`, `SCC`, `Cycle`, `Shortest`, `Prim`, or `Kruskal` must be selected.

Features can have additional relations across the branches of feature models. These relations are called *Cross-Tree Constraints (CTCs)* and as well as hierarchical feature relations they are usually expressed and checked using propositional logic, for further details refer to [18]. Some examples of CTCs are shown textually in Figure 1. For instance, `Num requires Search` means that whenever feature `Num` is selected, feature `Search` must also be selected. As another example, `Prim excludes Kruskal` means that both features cannot be selected at the same time for any product. Now we present the basic definitions on which CIT for 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 `[sel,`$\overline{sel}$`]` where `fs.sel` and `fs.`$\overline{sel}$ are respectively the set of selected and not-selected features in a system part of a SPL. Let `FL` be a feature list, thus `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 $\mathcal{FS}^{fm}$.

GPL has 73 distinct valid feature sets. Table I shows 12 of such valid feature sets, where selected features are ticked (✓) and unselected features are empty. An example of a valid feature set is `fs0` that computes the algorithms `Cycle`, `Num`, and `SCC` on `Directed` graphs using `DFS` search. Thus, the selected features are `fs0.sel`={`GPL, Driver, GraphType, Search, Algorithms, Benchmark, Directed, DFS, Num, SCC, Cycle`}, and the unselected features `fs0.`$\overline{sel}$={`Weight, Undirected, BFS, CC, Shortest, Prim, Kruskal`}. Consider now another feature set `gs` with selected features `SCC` and `Undirected`, that is, {`SCC,Undirected`} $\subset$ `gs.sel`. This feature set is invalid because these features violate the CTC that establishes that whenever feature `SCC` is selected then feature `Directed` must be selected, i.e. `SCC requires Directed`.

III. Combinatorial Interaction Testing for Software Product Lines

When CIT is applied to SPLs, rather than testing the complete SPLs, the goal is to select a representative subset of products where interaction errors are likely to occur, based on a variability model [19]. In this section, we present the basic terminology of CIT for SPLs based on our previous work [20].

Definition 4: **t-set**. *A t-set ts is a 2-tuple `[sel,`*$\overline{sel}$*`]` representing a partially configured product, defining the selection*

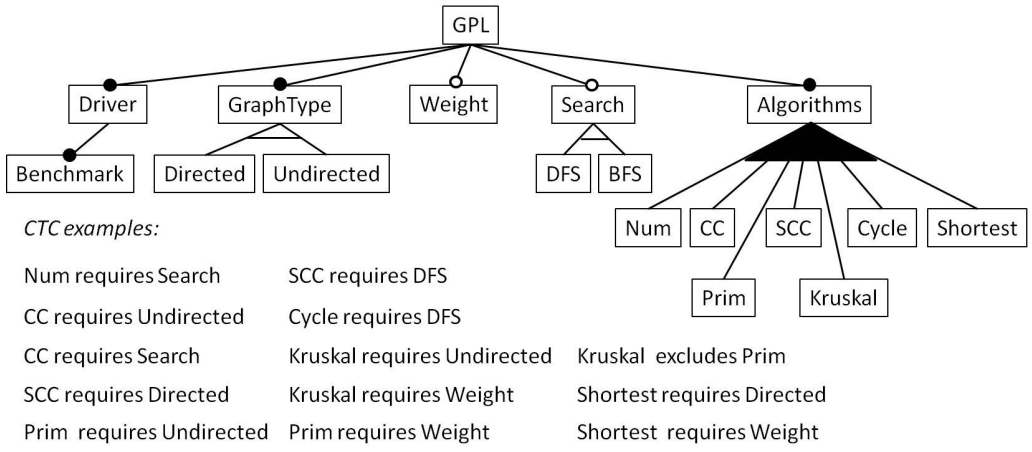

Fig. 1: Graph Product Line Feature Model [17].

TABLE I: 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	✓	✓	✓	✓		✓	✓		✓									✓
fs8	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		
fs9	✓	✓	✓	✓		✓	✓		✓								✓	
fs10	✓	✓	✓	✓	✓	✓	✓		✓		✓						✓	
fs11	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓				✓		

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

of t *features of the feature list FL, i.e.* $ts.sel \cup ts.\overline{sel} \subseteq FL \wedge ts.sel \cap ts.\overline{sel} = \emptyset \wedge |ts.sel \cup ts.\overline{sel}| = t$. *We say t-set* `ts` *is* `covered` *by feature set* `fs` *iff* $ts.sel \subseteq fs.sel \wedge ts.\overline{sel} \subseteq fs.\overline{sel}$.

Definition 5: **Valid t-set**. *A t-set* `ts` *is valid in a feature model* `fm` *if there exists a valid feature set* `fs` *that covers* `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* `tCA` *for a feature model* `fm` *is a set of valid feature sets that covers all valid t-sets in* $\mathcal{VTS}^{fm}$. *Formally,* $tCA \subseteq \mathcal{P}(FS^{fm})$ *where* $\forall ts \in VTS^{fm}, \exists fs \in tCA$ *such that* fs *covers* ts.

We now illustrate these concepts for `t=2` or *pairwise testing*. From the feature model in Figure 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 `[{Prim,Weight}, ∅]` is valid because there is at least one feature set, for instance `fs2` in Table I, where both features are selected. The 2-set `[∅, {Kruskal,Prim}]` is also valid because there is at least one valid feature set that does not have any of these features selected, for example feature set `fs0`. 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. GPL has 418 valid 2-sets, therefore a 2-wise covering array must contain all these pairs covered by at least one feature set. Table I is an example of a covering array for GPL [21].

IV. Systematic Mapping Study

Evidence-Based Software Engineering (EBSE) is a young software engineering research area whose main goal is *"to provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software"* [22]. Systematic mapping studies are one of the approaches championed by EBSE. Their purpose is to provide an overview of the available research and results within an area of knowledge, and categorize them according to different criteria such as type, frequency, publication forum, etc. [11]. In our mapping study, we followed the protocol proposed by Petersen et al. [11]. It consists of five processes

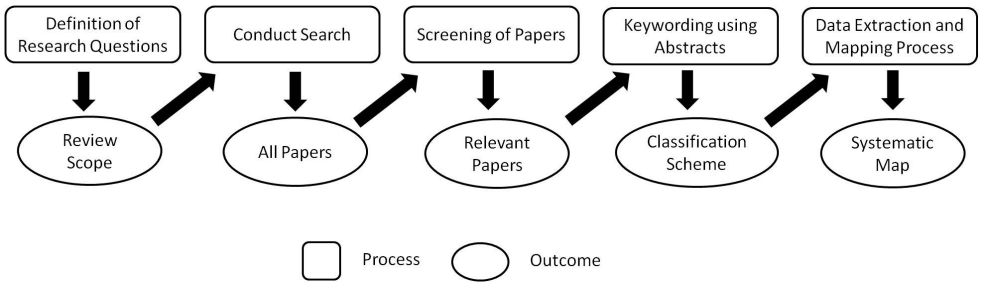

Fig. 2: Systematic Mapping Study Process [11].

shown in Figure 2. Next we describe each of the processes and how they were performed for our mapping study. In Section V we present the results obtained, and in Section VI their analysis.

A. Definition of Research Questions

Recall that the goal of our mapping study is to gather information and characterize the existing work on CIT applied to SPLs. Our mapping study focuses on the following research questions:

- ***RQ1.What are the techniques that have been used for CIT in SPLs?***
  Rationale: There has been a large number of techniques and algorithms developed for CIT that have been surveyed and analyzed (e.g. [9]), however their application focusing to SPL testing has not been collectively mapped and studied. Comparing and contrasting the CIT techniques applied to SPLs with those applied for general software systems could shed light on potential research avenues for further exploration.
- ***RQ2. What phases of CIT have been explored for SPLs?***
  Rationale: Recent work by Yilmaz et al. divides CIT approaches in two big phases [6]: *i) what phase* whose purpose is to select a group of products for testing, and *ii) how phase* whose purpose is to perform the test on the selected products. Our objective with this research question is to categorize the existing literature according to these two phases and to detect any possible research gaps that could be further addressed.
- ***RQ3. What are the case studies used for evaluation of the CIT approaches applied to SPLs and what is their provenance?***
  Rationale: From our previous experience in the subject [23], we observed that CIT approaches for SPLs employ a range of different case studies that come from academic, open source, or industrial sources. The goal of this question is to gather information regarding case study use for identifying common case studies that could be used to develop a benchmark for evaluation. Additionally, the provenance information can help us determining the maturity of the approaches, as industrial or open source examples tend to be more demanding and challenging.
- ***RQ4. What are the publication fora used?***
  Rationale: Identifying publication fora can help researchers and practitioners keep abreast with developments in the area as well as target future publications.

B. Conduct Search for Primary Sources

In this step of the systematic mapping study the search terms are defined. We selected two sets of terms, one for SPLs and a second one for CIT that cover the basic concepts found in the corresponding research subjects. The terms are shown in Table II.

TABLE II: Summary of SPL and CIT Search Terms.

SPL terms: software family, product family, software product family, software product line, SPL, product line, feature model, feature diagram, variability modeling
CIT terms: combinatorial interaction testing, CIT, combinatorial testing, pairwise testing, n-wise testing, interaction testing, covering array, t-way testing

The search was performed in the following two stages:

- *Publishing companies and general search engines*. We utilized the search engines provided by ScienceDirect, IEEExplore, ACM Digital Library, SpringerLink, and Google Scholar. These engines cover the main publication venues of the specialized journals, conferences, and workshops in both SPLs and general software engineering. Google Scholar was also used to search other publication outlets such as technical reports and dissertations.
- *Snowballing readings*. This stage implies using the reference list or citations of the identified papers to further search other sources [12], [24]. Additionally, we followed the guidelines proposed by Wohlin et al. that considers what other publications the identified papers are referenced by [24]. This stage was performed manually following the citation links provided by the publishing companies and Google Scholar.

The queries we performed took all the combinations of SPL terms and CIT terms. The searches included the title, abstract, and keywords of the papers, and additionally their contents whenever supported by the search engine. The following

fragment of a query is an example used in the IEEExplore engine[1]:

```
("product line") AND ("combinatorial
interaction testing" OR "CIT" OR
"combinatorial testing" OR "pairwise
testing" OR "n-wise testing" OR
"interaction testing" OR "covering
array" OR "t-way testing")
```

C. Screening of Papers for Inclusion and Exclusion

We looked for the search terms in the title, abstract and keywords. When necessary we also looked for details at the introduction or other places of the papers such as the evaluation section. The only criteria for inclusion in our mapping study was that a clear application of CIT techniques to SPL testing was presented. For exclusion criteria, we did not consider publications whose SPL testing techniques that bore no relation to CIT or were not written in English.

D. Keywording using Abstracts — Classification Scheme

We classified our articles into four dimensions that correspond to each question our mapping study addresses.

1) CIT techniques classification: For this dimension we consider each type of technique or algorithm used by the CIT approaches. Some examples of such techniques are constraint programming, evolutionary computation, or greedy algorithms [25], [26], [27]. It should be pointed out that a paper can rely on more than one technique to implement CIT or use multiple techniques for comparison. Our mapping study considers these two cases. If one category may subsume another we report the most specific one for classification[2].

2) CIT phases classification: As mentioned in Section IV-A, our mapping study classifies the articles into the two main phases of CIT (the *what* and the *how* phases) according to the recent work of Yilmaz et al. [6].

3) Case studies and their provenance classification: For this classification we considered the type of variability model used for expressing the valid combinations of products of the SPL from which a subset is selected for testing. Examples of variability models are feature models or constraint models. Regarding provenance, we use the following categories:

- *SPLOT*[3] which is a repository for feature models widely used within the SPL research community.
- *Random* when the variability model of a case study is generated randomly.
- *Open source* when case studies come from open source projects.
- *Academic* when the case studies come from academia, either from research papers or projects mostly carried out at universities or research institutions.
- *Industrial* when the case studies are part of actual industrial applications.
- *Undefined* when it was not possible to ascertain the provenance of a case study.
- *None* when no case studies were used for evaluation.

For each case study we also recorded the availability of the variability model that describes its combinations of products (e.g. feature model) and the actual implementation of the case study (i.e. source code and other artifacts needed for their execution).

4) Type of publication fora classification: The classification of publication fora was straightforward because we used the name of the journal, conference, workshop or book where the publication appeared.

E. Data Extraction and Mapping Study

We followed the next steps for gathering the data of our mapping study:

1) We created a guideline document defining each of the classification terms and an spreadsheet to collect the classification information. The spreadsheet contained the following data fields: *i)* CIT techniques used, *ii)* rationale for the categorization if any needed, *iii)* CIT phases addressed, *iv)* rationale for the categorization if any needed, *v)* type of variability model(s), *vi)* availability of variability model(s), *vii)* availability of implementation artifacts, *viii)* provenance of the case study(ies), *ix)* rationale for the categorization if any needed, *x)* a general field for any other remarks.
2) We formed two groups to carry out the classification task independently.
3) We held a meeting to pilot the classification terms. For this meeting each group had independently collected the data for some selected primary sources. The results obtained were compared and contrasted, and all the discrepancies were analyzed and clarified.
4) The two teams performed the classification of all primary sources independently.
5) We held a second meeting to discuss the classification for every single paper for each criterion.

V. Results

In this section we present and describe the results obtained in our systematic mapping study. As mentioned in Section IV-B, we first performed search queries in specialized repositories and search engines. These queries were performed between 11th December 2014 and 17th December 2014 and produced a total of 1,631 hits. As a second step we sieved the articles based on the title, abstract and keywords, resulting in 38 papers. For the third step, we looked into the introduction and other relevant parts of the papers. This resulted in the exclusion of 1 paper. The reason for excluding this paper was, that it was not in an SPL context. For the fourth step we performed snowballing readings, resulting in 10 more papers. The selection of these readings followed the same screening process of the third step. The four steps and their results are summarized in Figure 3. Appendix A lists the 47 articles that

[1]Some search queries had to be broken down into smaller queries because of the limitations of the search engines. Nonetheless our queries considered all possible combinations of SPL and CIT search terms.

[2]For example, if a paper uses an algorithm such as SPEA2, it is classified as multi-objective evolutionary algorithm but not as a genetic algorithm.

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

form the primary sources of our mapping study presented in the order they were found.

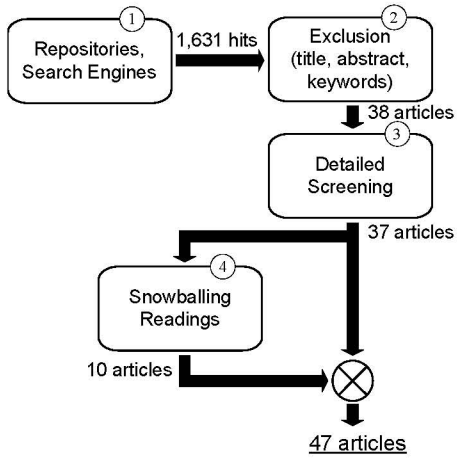

Fig. 3: Steps of search and primary sources selection

The first interesting result of our mapping study is the growth in number of publications over time as shown in Figure 4. From 2006 to 2010 we observed a low number of publications, followed by a sharp increase since 2011. The peak of publications was in 2013 where the number of publications doubled those from 2012. For 2014 the number of publications seems to decrease. However this could be because the digital libraries where still missing some of the recent publications. The latest paper is to be published in a journal in 2015, but was already online available at the time we performed our search.

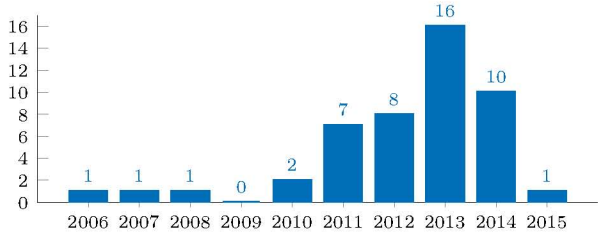

Fig. 4: Publications per year

A. Results RQ1 – Techniques Usage

Table III shows the CIT techniques found by our mapping study, the acronyms we use to refer to them, and the primary sources for each technique. We introduced a technique category *Generic* for techniques not specific to a particular CIT technique, but that can be used for any approach. For instance visualization (e.g. [S24]), reduction of t-wise tests, benchmark outline [S33], etc. Notice that a publication can employ more than one CIT technique, for instance S14 uses GRE and SA.

The first thing to notice in Table III is the large and diverse set of CIT techniques used, making a total of 13. The most used technique was GRE (GREedy algorithms), with 13 publications. Following this came CP (Constraint Programming) with 8 publications. The third most frequent technique was GE (GEneric) with 6 publications, followed by SA (Simulated Annealing), GA (Genetic Algorithm), EA (Evolutionary Algorithm), MOEA (Multi-Objective Evolutionary Algorithm) and STA (Static Analysis). The remaining references were spread out among the remaining 5 techniques.

TABLE III: Primary Sources and CIT Techniques

Technique	Acronym	Primary Sources Identifiers
Constraint Handling	CH	S21
Constraint Programming	CP	S1, S3, S13, S18, S25, S26, S30, S37
Evolutionary Algorithm (1+1)	EA	S10, S32, S35
Exact Multi-Objective Algorithm	EMOA	S19
Generic	GE	S2, S11, S24, S27, S33, S36
Genetic Algorithm	GA	S16, S40, S45, S47
Greedy algorithm	GRE	S4, S5, S8, S14, S15, S17, S22, S23, S28, S29, S31, S34, S38
Model-Based	MB	S43
Multi-Objective Evolutionary Algorithm	MOEA	S20, S44, S46
Random Search	RS	S41
Simulated Annealing	SA	S14, S23, S39, S42
Static Analysis	STA	S7, S9, S12
Statistical Test	ST	S6

TABLE IV: Primary Sources and CIT Phase

CIT Phase	Primary Sources Identifiers
What	S1, S2, S3, S4, S5, S6, S8, S9 S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44, S45, S46
How	S1, S3, S7, S9, S13, S14, S18, S22, S31, S45, S47

B. Results RQ2 – CIT Phases

Table IV lists the primary sources associated with the CIT phases described in Section IV-A. The *what phase* represents the selection of products to test, that represent the entire SPL. The *how phase* is concerned with the execution of tests and also using information from the test runs to refine the selection or order of the products to test (e.g. code coverage).

It is quite remarkable that the majority of publications were categorized in the first phase (i.e. *what phase*), 45 publications. Only 11 publications focus on the *how phase* of CIT, whereby 9 of them cover both phases. This leaves only two publications that have an exclusive focus on the *how phase*, one by Kim et al. [S7], who reduce the number of products to test during test execution and one by Wang et al. [S47] on test prioritization. In Section VI we analyze and discuss these findings.

C. Results RQ3 – Case Studies and their Provenance

In total the 47 publications found in our study contain 170 different case studies. Table V list the 12 most frequently used ones. The table shows the name of the case study, its provenance (P), the primary sources that included it and if a variability model (VM) or the source code are available.

The first thing to notice here is that most of the case studies are from the SPLOT repository. This is representative for all the publications, because most of the case studies where

TABLE V: Primary Sources and Case Study Provenance

Case Study	P	Primary Sources Identifiers	VM	Code
Linux Kernel	O	S8, S10, S5, S34, S42	✓	✓
Violet	O	S2, S5, S16, S34	✓	✓
Arcade Game	S	S2, S23, S30, S37	✓	
Coche Ecologico	S	S2, S21, S23, S30, S32, S35, S37, S41, S46	✓	
Electronic Shopping	S	S5, S21, S22, S23, S29, S30, S34, S37, S41	✓	
Model Transformation	S	S2, S22, S30, S37, S41, S46	✓	
Printers	S	S2, S21, S22, S32, S35, S41	✓	
Sienna	S	S23, S30, S37, S40	✓	
Smart Home	S	S5, S14, S30, S34, S37, S44, S46	✓	
Video Player	S	S21, S22, S32, S35, S41, S46	✓	
Web portal	S	S22, S23, S30, S37	✓	
Eclipse	O	S8, S15, S31, S34		✓

O = Open Source, S = SPLOT

taken from SPLOT or a similar repository. However, for most cases there is no source code available. The second most used type of case studies are open source projects. Here the source code is available, but it can be hard to find a variability model for these systems. Five of our primary sources used randomly generated variability models, and only one paper did not clearly define the provenance of the used case studies (i.e. *Undefined*). Furthermore 4 publications did not have case studies at all, since they did not conduct an experiment (i.e. *None*). Finally 11 publications included industrial case studies and 11 academic ones. For these types it is usually very hard or even impossible to get access to the variability model or source code.

The majority of our sources, 39 papers, use feature models as variability models. Four papers use sets of constraints, two use a Orthogonal Variability Model and two use a Multi-Perspective Feature Model respectively.

D. Results RQ4 – Publication Fora

We divided the publications in three groups: conferences, journals, and miscellaneous. The last group contains workshop papers, technical reports, and dissertations. The division was made to reflect length, nature and maturity of the sources analyzed. Our study found 14 conference fora, 3 journal fora, and 9 miscellaneous. Table VI shows the primary sources categorized along these three groups.

As one could expect the most frequent venues for publications are the most prominent conferences and workshops for SPL, respectively SPLC with 7 publications and Vamos with 5. These were followed with 3 publications by TSE, which makes it the most frequent journal in our sources. ICSTW (a specialized workshop in software testing) also has 3 publications. Followed by MODELS and SPLCW with 2 publications each. The remaining publications were more or less evenly spread out.

In summary out of the 47 primary sources, 24 correspond to conference fora, 5 to journal publications, and 18 to miscellaneous publications. We believe these numbers denote on one hand a clear interest among several communities in the research at the intersection of CIT on SPLs, but on the other hand they also highlight that this research area is still very young (e.g. very few journal publications and many workshop publications).

VI. Analysis

In this section we analyze the findings revealed by our systematic mapping study. We refer to some of the relevant primary sources and shortly discuss open questions and potential areas for further research.

A. CIT Phases

As discussed in Section IV-A we distinguish CIT in two phases, the *what phase* and the *how phase*. However we observed that the majority of publications to date focus on the *what phase*. Though we believe that there are possibilities for investigation in both phases, the *what phase* seems to be better researched already, which means there a multiple areas open for further research on the *how phase*, such as exploiting variability knowledge at run time as done by Kim et al. who use run time information to reduce the number of products. They analyze which code is executed by which tests and check if other products have any difference along the execution path. If not, it is assumed that they do not have to be tested [S7].

B. Multi-Objective Optimization

Quite often SPL problems require the optimization of multiple and sometimes contradicting objectives. For example the minimization of the sizes of SPL test suites and the maximization of their t-wise coverage. The work by Lopez-Herrejon et al. proposes an exact algorithm that computes the true Pareto front of feature models using SAT solvers and presents scalability issues for larger feature models [S19]. Lopez-Herrejon et al. also provided a comparison of four classical multi-objective evolutionary algorithms (i.e. NSGA-II, PAES, MOCell, and SPEA2) for the computation of pairwise testing and analyzed three different seeding strategies for the initial population [S20]. There are only two publications that use multi-objective optimization in our primary sources. This leaves possibilities for further investigations using other multi-objective algorithms, including more optimization objectives that consider information such as control-flow.

C. Definition of a Community-Wide Benchmark

Our systematic mapping study found that feature models from the SPLOT repository were the most common type of variability model used for CIT on SPLs. However, the selection of which feature models to analyze in each paper seemed to be arbitrary, at worst, or partially-justified, at best. A community-wide benchmark could effectively help to objectively compare different CIT techniques. Such a benchmark should not only provide both the variability model and source code or artifacts, but also a set of metrics beyond covering array size or execution time. The importance of such a benchmark has already been expressed by Cohen et al. in [S36], and further was advocated by Lopez-Herrejon et al. [S33] who provide a first list of candidate case studies.

TABLE VI: Publication Fora

Acronym	Primary Sources Identifiers	Publication Name
	Conference Publications	
AOSD	S9	International Conference on Aspect-Oriented Software Development
ASE	S12	International Conference on Automated Software Engineering
CAISE	S40	Conference on Advanced Information Systems Engineering
CEC	S20	IEEE Congress on Evolutionary Computation
EuroPLoP	S8	European Conference on Pattern Languages of Programs
ESEC/FSE	S7	Joint Meeting - European Software Engineering Conference and the Symposium on the Foundations of Software Engineering
GECCO	S16, S44	Genetic and Evolutionary Computation Conference
HASE	S21	International IEEE Symposium on High-Assurance Systems Engineering
ICSM	S19	International Conference on Software Maintenance
ICST	S22, S43	International Conference on Software Testing, Verification and Validation
ICTSS	S31	International Conference on Testing Software and Systems
ISSRE	S37	International Symposium on Software Reliability Engineering
ISSTA	S36	International Symposium on Software Testing and Analysis
MODELS	S5, S15	International Conference on Model Driven Engineering Languages and Systems
SPLC	S13, S14, S17, S30, S45, S46, S47	International Conference Software Product Lines
	Journal Publications	
ESE	S39	Empirical Software Engineering
JSS	S28	Journal of Systems and Software
TSE	S27, S35, S38	IEEE Transactions on Software Engineering
	Miscellaneous Publications	
CoRR	S32, S33, S42	Computing Research Repository
ICSTW	S23, S25, S41	Workshop International Conference on Software Testing, Verification and Validation
PhdThesis	S34	PHD thesis
PLEASE	S26	International Workshop on Product Line Approaches in Software Engineering
ROSATEA	S11	Workshop on Role of Software Architecture for Testing and Analysis
SPLCW	S1, S10	International Conference Software Product Lines Workshop Proceedings
Vamos	S2, S3, S4, S6, S18	International Workshop on Variability Modelling of Software-intensive Systems
VariComp	S29	International Workshop on Variability and Composition
VISSOFT	S24	Working Conference on Software Visualization

VII. Threats to Validity

We face similar validity threats as any other systematic mapping study. A first threat to validity is related to the selection of paper repositories and the search queries applied to them. To address this threat we used the major bibliography search engines and carefully chose a selection of search terms covering SPL and CIT. A second threat is related to the classification scheme for CIT phases. We used the phases defined by Yilmaz et al. [6] for our classification, to adress this threat. A third threat to validity is related to the selection of criteria to include and exclude publications. In this case the only criteria was that a clear application of a CIT approach to SPL testing was presented. To give a more comprehensive view of the area we included all types of publications, from non-peer-reviewed to journal publications. A fourth threat is related to the extraction of data for the classification. To address this threat, we carried out the classification using commonly-agreed terms, that were piloted and used by two independent groups. Subsequently we held a meeting discussing our classification results until a consensus was reached.

VIII. Related Work

There exists an extensive body of literature on CIT and SPL testing. In this section we briefly describe the salient studies and surveys on both subjects.

CIT studies. Grindal et al. surveyed 16 different combination strategies along which they classified salient research literature [7]. Furthermore, they proposed a hierarchy for coverage criteria. Nie and Leung performed a general survey in CIT testing [8]. Their work provides an overview of the evolution of CIT research and a classification scheme for state of the art. Additionally, they put forward a set of directions for future research on CIT. Similarly, Ahmed and Zamli perfomed a short review that focused on the computation of covering arrays and illustrate some of their applications [28]. A most recent and extensive survey has been presented by Khalsa and Labiche who performed an analysis of 75 CIT algorithms and tools [9]. Their work considers factors such as type of techniques, generation strategies, selection criteria, and coverage strength. In contrast with our work, none of these surveys address the domain of SPL testing.

SPL testing studies. Two contemporary systematic mapping studies in SPL testing [29], [30], provide a wide overview of the area. They focus on categorizing SPL approaches along criteria within the domain of SPL such as handling of variability and variant binding times, as well as other aspects like test organization and process. A recent literature review categorized SPL testing strategies into two fundamental aspects: the selection of the products to test, and the actual testing of the products [31]. Despite the extensive work carried out on both of these aspects, the authors found that there is still a great lack of empirical industrial applications. This latter finding also corroborates the importance of proposing a benchmark (albeit not with industrial applications) on which to measure and compare different CIT approaches applied to SPLs. In contrast with these studies, our work focuses on identifying the techniques, algorithms, and tools used for CIT in SPLs as well as cataloguing the case studies used for evaluation with the goal of identifying common examples to constitute the basis of a benchmark.

IX. Conclusions and Future Work

In this paper we present the results of our mapping study on applications of CIT techniques on SPLs. Our study corroborated the increasing interest in applying CIT techniques to SPLs, as was shown by the increasing number of publications over the last years. The majority of publications focus on

the *what phase* of CIT. The most common technique used is greedy algorithms. Our work revealed opportunities for future research, for example, in developing community-wide testing benchmarks, leveraging more information with multi-objective optimization, and investigating in ways to better utilize the *how phase*. We hope this mapping study not only serves to highlight the main research topics on the subject, but also inspires researchers and practitioners to explore subjects at the intersection of both research communities. Part of our future work is to perform a more in depth analysis on the primary sources our study identified, for instance, why there has been more focus on the *what phase*, and why are some techniques more frequently used, and if there are any common techniques representing constraints, and what is the status of the tool support in this area.

ACKNOWLEDGMENT

This work has been supported by the competence centers programme COMET - Competence Centers for Excellent Technologies of the Austrian Research Promotion Agency (FFG) and the Austrian Science Fund (FWF) project P25289.

REFERENCES

[1] D. S. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling step-wise refinement," *IEEE Trans. Software Eng.*, vol. 30, no. 6, pp. 355–371, 2004.

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

[3] F. J. van d. Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[4] T. Käkölä and J. C. Dueñas, Eds., *Software Product Lines - Research Issues in Engineering and Management*. Springer, 2006.

[5] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques," *Softw., Pract. Exper.*, vol. 35, no. 8, pp. 705–754, 2005.

[6] C. Yilmaz, S. Fouché, M. B. Cohen, A. A. Porter, G. Demiröz, and U. Koc, "Moving forward with combinatorial interaction testing," *IEEE Computer*, vol. 47, no. 2, pp. 37–45, 2014.

[7] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: a survey," *Softw. Test., Verif. Reliab.*, vol. 15, no. 3, pp. 167–199, 2005.

[8] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, p. 11, 2011.

[9] S. K. Khalsa and Y. Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," Department of Systems and Computer Engineering Carleton University, Tech. Rep., 2014.

[10] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering. version 2.3." EBSE Technical Report EBSE-2007-01, 2007.

[11] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *EASE*. British Computer Society, 2008, pp. 68–77.

[12] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using Mapping Studies in Software Engineering," in *PPIG*. Lancaster University, 2008, pp. 195–204.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012.

[14] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research - a participant-observer case study," *Information & Software Technology*, vol. 53, no. 6, pp. 638–651, 2011.

[15] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski, "Cool features and tough decisions: a comparison of variability modeling approaches," in *VaMoS*, U. W. Eisenecker, S. Apel, and S. Gnesi, Eds. ACM, 2012, pp. 173–182.

[16] 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., 1990.

[17] 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.

[18] 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.

[19] 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.

[20] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Using feature model knowledge to speed up the generation of covering arrays," in *VaMoS*, 2013, p. 16.

[21] R. E. Lopez-Herrejon, J. F. Chicano, J. Ferrer, A. Egyed, and E. Alba, "Multi-objective optimal test suite computation for software product line pairwise testing," in *ICSM*. IEEE, 2013, pp. 404–407.

[22] B. Kitchenham, T. Dybaa, and M. Jorgensen, "Evidence-based software engineering," in *ICSE*. IEEE CS Press, 2004, pp. 273–281.

[23] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, L. Linsbauer, A. Egyed, and E. Alba, "A hitchhiker's guide to search-based software engineering for software product lines," *CoRR*, vol. abs/1406.2823, 2014.

[24] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *EASE*, M. J. Shepperd, T. Hall, and I. Myrtveit, Eds. ACM, 2014, p. 38.

[25] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[26] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[27] S. Luke, *Essentials of Metaheuristics*. Lulu, 2009, http://cs.gmu.edu/∼sean/book/metaheuristics/.

[28] B. S. Ahmed and K. Z. Zamli, "A review of covering arrays and their application to software testing," *Journal of Computer Science*, vol. 7, no. 9, pp. 1375–1385, 2011.

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

[30] 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.

[31] I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. de Almeida, "On strategies for testing software product lines: A systematic literature review," *Information and Software Technology*, vol. 56, no. 10, pp. 1183 – 1199, 2014.

APPENDIX

PRIMARY SOURCES LIST

[S1] Sebastian Oster, Marius Zink, Malte Lochau, and Mark Grechanik. Pairwise feature-interaction testing for spls: potentials and limitations. In *SPLC Workshops*, page 6, 2011.

[S2] Evelyn Nicole Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. Using feature model knowledge to speed up the generation of covering arrays. In *VaMoS*, page 16, 2013.

[S3] Sebastian Oster, Ivan Zorcic, Florian Markert, and Malte Lochau. Moso-polite: tool support for pairwise and model-based software product line testing. In *VaMoS*, pages 79–82, 2011.

[S4] Ana B. Sánchez, Sergio Segura, and Antonio Ruiz Cortés. The drupal framework: a case study to evaluate variability testing techniques. In *VaMoS*, page 11, 2014.

[S5] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. Properties of realistic feature models make combinatorial testing of product lines feasible. In *MODELS*, pages 638–652, 2011.

[S6] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Pierre-Yves Schobbens, Axel Legay, and Patrick Heymans. Towards statistical prioritization for software product lines testing. In *VaMoS*, 2014.

[S7] Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don S. Batory, Sabrina Souto, Paulo Barros, and Marcelo d'Amorim. Splat: lightweight dynamic analysis for reducing combinatorics in testing configurable systems. In *ESEC/FSE*, pages 257–267, 2013.

[S8] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. Bow tie testing: a testing pattern for product lines. In *EuroPLoP*, page 9, 2011.

[S9] Chang Hwan Peter Kim, Don S. Batory, and Sarfraz Khurshid. Reducing combinatorics in testing product lines. In *AOSD*, pages 57–68, 2011.

[S10] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Pledge: a product line editor and test generation tool. In *SPLC Workshops*, pages 126–129, 2013.

[S11] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Coverage and adequacy in software product line testing. In *ROSATEA*, pages 53–63, 2006.

[S12] Chang Hwan Peter Kim, Don S. Batory, and Sarfraz Khurshid. Eliminating products to test in a software product line. In *ASE*, pages 139–142, 2010.

[S13] Dusica Marijan, Arnaud Gotlieb, Sagar Sen, and Aymeric Hervieu. Practical pairwise testing for software product lines. In *SPLC*, pages 227–235, 2013.

[S14] Mustafa Al-Hajjaji, Thomas Thum, Jens Meinicke, Malte Lochau, and Gunter Saake. Similarity-based prioritization in software product-line testing. In *SPLC 14*, 2014.

[S15] Martin Fagereng Johansen, Øystein Haugen, Franck Fleurey, Anne Grete Eldegard, and Torbjørn Syversen. Generating better partial covering arrays by modeling weights on sub-product lines. In *MoDELS*, pages 269–284, 2012.

[S16] Roberto Erick Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Evelyn Nicole Haslinger, Alexander Egyed, and Enrique Alba. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In *GECCO*, pages 1255–1262, 2014.

[S17] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *SPLC (1)*, pages 46–55, 2012.

[S18] Michaela Steffens, Sebastian Oster, Malte Lochau, and Thomas Fogdal. Industrial evaluation of pairwise SPL testing with moso-polite. In *VaMoS*, pages 55–62, 2012.

[S19] Roberto E. Lopez-Herrejon, Francisco Chicano, Javier Ferrer, Alexander Egyed, and Enrique Alba. Multi-objective optimal test suite computation for software product line pairwise testing. In *ICSM*, pages 404–407, 2013.

[S20] Roberto Erick Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Alexander Egyed, and Enrique Alba. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In *CEC*, pages 387–396, 2014.

[S21] Linbin Yu, Feng Duan, Yu Lei, Raghu Kacker, and D. Richard Kuhn. Combinatorial test generation for software product lines using minimum invalid tuples. In *HASE*, pages 65–72, 2014.

[S22] Ana B. Sánchez, Sergio Segura, and Antonio Ruiz Cortés. A comparison of test case prioritization criteria for software product lines. In *ICST*, pages 41–50, 2014.

[S23] Andrea Calvagna, Angelo Gargantini, and Paolo Vavassori. Combinatorial testing for feature models using citlab. In *ICST Workshops*, pages 338–347, 2013.

[S24] Roberto E. Lopez-Herrejon and Alexander Egyed. Towards interactive visualization support for pairwise testing software product lines. In *VISSOFT*, pages 1–4, 2013.

[S25] Sachin Patel, Priya Gupta, and Vipul Shah. Combinatorial interaction testing with multi-perspective feature models. In *ICST*, pages 321–330, 2013.

[S26] Sachin Patel, Priya Gupta, and Vipul Shah. Feature interaction testing of variability intensive systems. In *PLEASE*, pages 53–56, 2013.

[S27] Andrea Arcuri and Lionel C. Briand. Formal analysis of the probability of interaction fault detection using random testing. *IEEE Trans. Software Eng.*, 38(5):1088–1099, 2012.

[S28] Beatriz Pérez Lamancha, Macario Polo, and Mario Piattini. Prow: A pairwise algorithm with constraints, order and weight. *ScienceDirect The Journal of Systems and Software (JSS)*, 99:1–19, 2015.

[S29] Matthias Kowal, Sandro Schulze, and Ina Schaefer. Towards efficient spl testing by variant reduction. In *VariComp*, pages 1–5, 2013.

[S30] Sebastian Oster, Florian Markert, and Philipp Ritter. Automated incremental pairwise testing of software product lines. In *SPLC*, pages 196–210, 2010.

[S31] Martin Fagereng Johansen, Øystein Haugen, Franck Fleurey, Erik Carlson, Jan Endresen, and Tormod Wien. A technique for agile and automatic interaction testing for product lines. In *ICTSS*, pages 39–54, 2012.

[S32] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines. *CoRR*, abs/1211.5451, 2012.

[S33] Roberto E. Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Evelyn Nicole Haslinger, Alexander Egyed, and Enrique Alba. Towards a benchmark and a comparison framework for combinatorial interaction testing of software product lines. *CoRR*, abs/1401.5367, 2014.

[S34] Martin Fagereng Johansen. *Testing Product Lines of Industrial Size: Advancements in Combinatorial Interaction Testing*. PhD thesis, University of Oslo, 2013.

[S35] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. Software Eng.*, 40(7):650–670, 2014.

[S36] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *ISSTA*, pages 129–139, 2007.

[S37] Aymeric Hervieu, Benoit Baudry, and Arnaud Gotlieb. PACOGEN: automatic generation of pairwise test configurations from feature models. In *ISSRE*, pages 120–129, 2011.

[S38] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. Software Eng.*, 34(5):633–650, 2008.

[S39] Brady J. Garvin, Myra B. Cohen, and Matthew B. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, 16(1):61–102, 2011.

[S40] Faezeh Ensan, Ebrahim Bagheri, and Dragan Gasevic. Evolutionary search-based test generation for software product line feature models. In *ICAiSE*, pages 613–628, 2012.

[S41] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. In *ICST Workshops*, pages 188–197, 2013.

[S42] Evelyn Nicole Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. Improving CASA runtime performance by exploiting basic feature model analysis. *CoRR*, abs/1311.7313, 2013.

[S43] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *ICST*, pages 459–468, 2010.

[S44] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *GECCO*, pages 1493–1500, 2013.

[S45] Zhihong Xu, Myra B. Cohen, Wayne Motycka, and Gregg Rothermel. Continuous test suite augmentation in software product lines. In *SPLC*, pages 52–61, 2013.

[S46] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Multi-objective test generation for software product lines. In *SPLC*, pages 62–71, 2013.

[S47] Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius Liaaen. Multi-objective test prioritization in software product line testing: an industrial case study. In *SPLC*, pages 32–41, 2014.