Multi-Objective Optimal Test Suite Computation for Software Product Line Pairwise Testing

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

***Abstract*—Software Product Lines (SPLs) are families of related software products, which usually provide a large number of feature combinations, a fact that poses a unique set of challenges for software testing. Recently, many SPL testing approaches have been proposed, among them pairwise combinatorial techniques that aim at selecting products to test based on the pairs of feature combinations such products provide. These approaches regard SPL testing as an optimization problem where either coverage (maximize) or test suite size (minimize) are considered as the main optimization objective. Instead, we take a multi-objective view where the two objectives are equally important. In this exploratory paper we propose a zero-one mathematical linear program for solving the multi-objective problem and present an algorithm to compute the true Pareto front, hence an optimal solution, from the feature model of a SPL. The evaluation with 118 feature models revealed an interesting trade-off between reducing the number of constraints in the linear program and the runtime which opens up several venues for future research.**

I. Introduction

Software Product Lines (SPLs) are families of related software products, where each product provides a unique combination of *features* (i.e. increments in program functionality [1]). Some of the benefits of SPLs are increased software reuse, faster product customization, and reduced time to market [2]. A *feature model (FM)* represents all the possible feature combinations (typically a large a number) available in an SPL. The number of combinations poses a unique set of challenges because testing each individual product may not be technically or economically feasible.

Recent surveys and mapping studies on SPL testing [3], [4], attest the increasing relevance of the topic within the SPL community. Salient among the SPL testing approaches are those based on *Combinatorial Interaction Testing (CIT)*, whose premise is to select a group of products where faults due to feature interactions are more likely to occur [5]. Here most of the focus has been on *pairwise* interactions, meaning that these techniques consider the four possible combinations between any two features[1]. The combination of features in a product of an SPL determines the set of pairwise feature combinations that the product *covers*. Pairwise SPL testing aims to select a set of products such that their feature combinations cover the possible combinations of *all* interactions between two features according to the feature model of the SPL. This set of products is called a *test suite*. Pairwise SPL testing approaches have used different techniques such as simulated annealing [6], evolutionary algorithms [7], and constraint programming [8]. These approaches regard SPL pairwise testing as an optimization problem where either coverage (maximize) or test suite size (minimize) are considered as the main optimization objective. Instead, we regard SPL pairwise testing as a multi-objective optimization problem where the two objectives, coverage and test suite size, are equally important. In the bi-objective formulation of the problem we say that one solution *dominates* another if the first is not worse than the second one in any objective and it is better in at least one objective. A set of solutions is said to be *non-dominated* if none dominates another. A *Pareto optimal set* is a set of non-dominated solutions each of which is not dominated by any other solution in the search space. The *Pareto front* is the projection of this set in the objective space, a plot containing the values of the objective functions for each solution. For more details on multi-objective optimization please refer to [9]. We present a zero-one mathematical linear program for solving the multi-objective problem and an algorithm that computes the true Pareto front of a feature model using SAT solvers. This front is the optimal solution for both objectives. We applied our approach to 118 publicly available feature models and were able to obtained their Pareto front. Our evaluation found a correlation between runtime and number of products in the feature model and revealed a trade-off between reducing the number of constraints in the mathematical linear program and runtime that we plan to explore as future work.

II. Feature Models and Running Example

Feature models have become the *de facto* standard for modelling the common and variable features of an SPL and their relationships collectively forming a tree-like structure. The nodes of the tree are the features which are depicted as labelled boxes, and the edges represent the relationships among them. Feature models denote the set of feature combinations that the products of an SPL can have [10]. Figure 1 shows the feature model of our running example, the *Graph Product Line (GPL)* [11], a standard SPL of basic graph algorithms that has been extensively used as a case study in the product line community. In GPL, a product is a collection of algorithms applied to directed or undirected graphs.

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

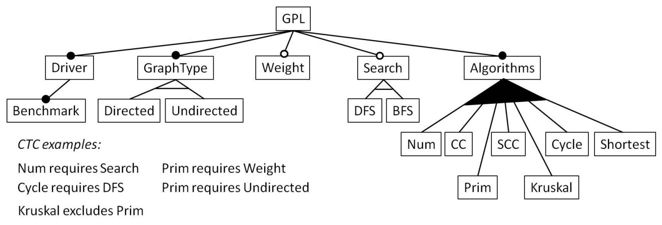

Fig. 1. Graph Product Line Feature Model

In a feature model, each feature (except the root) has one parent feature and can have a set of child features. Notice here that a child feature can only be included in a feature combination of a valid product if its parent is included as well. The root feature is always included. There are four kinds of feature relationships: *i) Mandatory features* are depicted with a filled circle. A mandatory feature is selected whenever its respective parent feature is selected. For example, features `Driver` and `GraphType`, *ii) 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 `Weight`, *iii) Exclusive-or relations* are depicted as empty arcs crossing over a set of 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 `Search` is selected, then either feature `DFS` or feature `BFS` must be selected, *iv) 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`, and `Kruskal` must be selected. Besides the parent-child relations, features can also relate across different branches of the feature model with the so called *Cross-Tree Constraints (CTC)*. Figure 1 shows some of the CTCs of our feature model[2]. For instance, `Cycle requires DFS` means that whenever feature `Cycle` is selected, feature `DFS` must also be selected. These constraints as well as those implied by the hierarchical relations between features are usually expressed and checked using propositional logic, for further details refer to [12].

Let us illustrate pairwise coverage in GPL. This example has 73 distinct products each with its unique feature combination. Consider for instance the product that computes numbering in DFS order on directed graphs without weight. For this product the features selected are: `GPL`, `Driver`, `Benchmark`, `GraphType`, `Directed`, `Search`, `DFS`, `Algorithms`, and `Num`. Some examples of combinations of pairs of feature interactions are: `GPL` and `Search` selected, `Weight` and `Undirected` not selected, `CC` not selected and `Driver` selected. An example of invalid pair, i.e. not denoted by the feature model, is features `Directed` and `Undirected` both selected. Notice that this pair is not valid because they are part of an exclusive-or relation. In total, GPL has 418 valid pairs, so a test suite for GPL must have these pairs covered by at least one product feature combination.

[2]In total, the feature model has 13 CTCs for further details refer to [11]

III. Mathematical Linear Program

We are interested in minimizing the number of test products and maximizing the pairwise coverage. Since we want to compute the Pareto front of the multi-objective optimization problem we proceed by fixing the number of test products and defining a zero-one mathematical program that maximizes coverage. The approach presented here relates to the work by Arito et al. [13] for solving a multi-objective test suite minimization problem in regression testing.

A zero-one program is an integer program in which the variables can only take values 0 or 1 [14]. The details of the algorithm applied are explained in Section IV. In this section we describe the zero-one program. Let us call n to the number of test products (that is fixed) and f to the number of features of the FM. We will use the set of decision variables $x_{i,j} \in \{0,1\}$ where $i \in \{1,2,\ldots,n\}$ and $j \in \{1,2,\ldots,f\}$. Variable $x_{i,j}$ is 1 if product i has feature j and 0 otherwise. Not all the combinations of features form valid products. Following [12], we can express the validity of any product in an FM as a boolean formula. These boolean formulas can be expressed in Conjunctive Normal Form (CNF) as a conjunction of clauses, which in turn can be expressed as constraints in a zero-one program. The way to do it is by adding one constraint for each clause in the CNF. Let us focus on one clause and let us define the Boolean vectors v and u as follows [15]:

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

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

With the help of u and v we can write the constraint that corresponds to one CNF clause for the i-th product as:

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

As an illustration, in the GPL model let us suppose that $Search$ is the 8-th feature and Num is the 12-th one. The cross-tree constraint "Num requires $Search$" can be written in CNF with the clause $\neg Num \vee Search$ and translated to a zero-one constraint as: $1 - x_{i,12} + x_{i,8} \geq 1$.

Our focus is pairwise coverage. This means that we want for each pair of features to cover 4 cases: both unselected, both selected, first selected and second unselected and vice versa. We introduce one variable in our program for each product, each pair of features and each of these four possibilities. The variables, called $c_{i,j,k,l}$, take value 1 if product 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; $l = 1$, second selected and first unselected; $l = 2$, 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 need to add the following constraints for all $i \in \{1, \dots, n\}$ and all $1 \le j < k \le f$:

$$2c_{i,j,k,0} \le (1 - x_{i,j}) + (1 - x_{i,k}) \le 1 + c_{i,j,k,0} \quad (2)$$
$$2c_{i,j,k,1} \le (1 - x_{i,j}) + x_{i,k} \le 1 + c_{i,j,k,1} \quad (3)$$
$$2c_{i,j,k,2} \le x_{i,j} + (1 - x_{i,k}) \le 1 + c_{i,j,k,2} \quad (4)$$
$$2c_{i,j,k,3} \le x_{i,j} + x_{i,k} \le 1 + c_{i,j,k,3} \quad (5)$$

Variables $c_{i,j,k,l}$ inform about the coverage in one product. We need new variables to count the pairs covered when all the products 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 $1 \le j < k \le f$ and $0 \le l \le 3$:

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

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. We can write this as:

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

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 products.

IV. Algorithm

The algorithm we use for obtaining the optimal Pareto set is given in Figure 1. This algorithm takes as input the FM and provides the optimal Pareto set. It starts by adding to the set two solutions that are always in the set: the empty solution (with zero coverage) and one arbitrary solution (with coverage C_2^f, number 2-combinations of the set of features). After that it enters a loop in which successive zero-one linear programs are generated for an increasing number of products starting at 2. Each mathematical model is solved using a extended SAT solver: `MiniSat+`[3]. This solver provides a test suite with the maximum coverage. This solution is stored in the optimal Pareto set. The algorithm stops when adding a new product to the test suite does not increase the coverage. The result is the optimal Pareto set.

V. Experiments

This section describes how the evaluation was carried and its scalability analysis. The experimental corpus of our evaluation is composed by a benchmark of 118 feature models, whose number of products ranges from 16 to 640 products, that are publicly available from the SPL Conqueror [16] and the SPLOT [17] repositories. The objectives to optimize are the number of products required to test the SPL and the achieved coverage. It is desirable to obtain a high value of coverage in a low number of products to test the SPL, so they are conflicting objectives. Additionally, as performance measure we have also analyzed the time required to run the algorithm, since we want the algorithm to be as fast as possible. For comparison purposes these experiments have been run in a cluster of 16 machines with Intel Core2 Quad processors Q9400 (1 core per experiment) at 2.66 GHz and 4 GB memory running Ubuntu 12.04.1 LTS and managed by the HT Condor 7.8.4 manager.

We computed the Pareto optimal front for each model. Figure 2 shows this front for our running example GPL, where the total coverage is obtained with 12 products, and for every test suite size the obtained coverage is also optimal. As our approach is able to compute the Pareto optimal front for every feature model in our corpus, it makes no sense to analyze the quality of the solutions. Instead, we consider more interesting to study the scalability of our approach. For that, we analyzed the execution time of the algorithm as a function of the number of products represented by the feature model as shown in Figure 3. In this figure we can observe a tendency: the higher the number of products, the higher the execution time. Although it cannot be clearly appreciated in the figure, the execution time does not grow linearly with the number of products, the growth is faster than linear.

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

Algorithm 1 Algorithm for obtaining the optimal Pareto set.

```
$optimal\_set \leftarrow \{\emptyset\}$;
$cov[0] \leftarrow 0$;
$cov[1] \leftarrow C_2^f$;
$sol \leftarrow$arbitraryValidSolution($fm$);
$i \leftarrow 1$;
while $cov[i] \neq cov[i-1]$ do
    $optimal\_set \leftarrow optimal\_set \cup \{sol\}$;
    $i \leftarrow i + 1$;
    $m \leftarrow$parepareMathModel($fm$,$i$);
    $sol \leftarrow$solveMathModel($m$);
    $cov[i] \leftarrow |sol|$;
end while
```

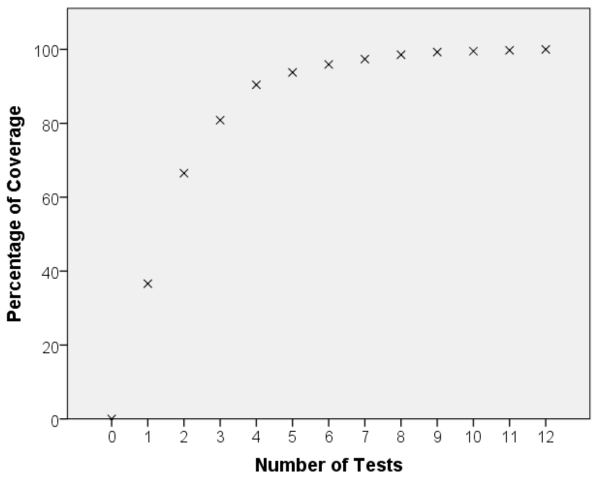

Fig. 2. Pareto optimal front for our running example (GPL).

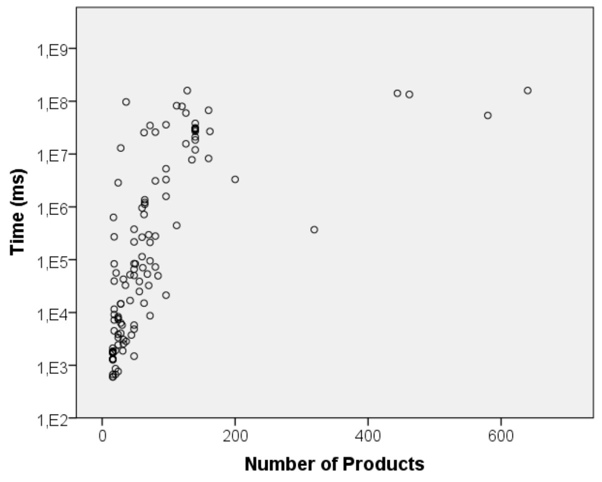

Fig. 3. Time (log scale) required to find optimal Pareto set against the number of products of the feature models.

In order to check our intuition, we have performed a Spearman's rank correlation test. This test's coefficient ρ takes into account the rank of the samples instead of the samples themselves. The correlation coefficient between the execution time and the number of products denoted by a feature model is 0.831. This is a very high value that confirms our expectations, the higher the number of products, the higher the execution time of the algorithm. We also computed the Spearman's rank correlation for the execution time against the number of features of the feature models which was quite lower (0.407). This is because two feature models with the same number of features could denote significantly different number of products depending on the constraints derived from the relationships between the features. In summary, the best indicator of the execution time of our approach is the number of products denoted by a feature model.

VI. Conclusions and Future Work

We have proposed an approach to exactly obtain the optimal Pareto set of the multi-objective SPL pairwise testing problem. We defined a zero-one linear mathematical program and an algorithm based on SAT solvers for obtaining the optimal Pareto set. By construction the solution obtained using this approach is optimal and could serve as reference for measuring the quality of the solutions proposed by approximated methods.

The evaluation revealed a generally large runtime for our feature models. This fact prompted us to analyze the impact of the number of products and number of features in runtime. We found a high correlation in the first case and a low correlation in the second case. As a result of this finding our future work is twofold. First, we want to streamline the mathematical program representation in order to reduce the runtime of the algorithm. We observed that some of the constraints can be redundant. For instance, features that are selected in all the products of the product line do not need a variable since they are valid for any product. Similarly, there are pairs of feature combinations, that is $c_{i,j,k,l}$ variables, that are not valid according to the feature model and hence can be eliminated [18]. We also noticed that removing some of the redundant constraints can increase the runtime, while adding more constraints could help the SAT solver search for a solution. We plan to study the right balance of both reducing and augmenting constraints. Second, we will look at larger feature models to further study the scalability of our approach.

Acknowledgements

This research is partially funded by the Austrian Science Fund (FWF) project P21321-N15 and Lise Meitner Fellowship M1421-N15, the Spanish Ministry of Economy and Competitiveness and FEDER under contract TIN2011-28194 and fellowship BES-2012-055967.

References

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

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

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

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

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

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

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

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

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

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

[11] 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.

[12] 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.

[13] F. Arito, F. Chicano, and E. Alba, "On the application of sat solvers to the test suite minimization problem," in *SSBSE*, ser. Lecture Notes in Computer Science, G. Fraser and J. T. de Souza, Eds., vol. 7515. Springer, 2012, pp. 45–59.

[14] L. A. Wolsey, *Integer Programming*. Wiley, 1998.

[15] A. M. Sutton, L. D. Whitley, and A. E. Howe, "A polynomial time computation of the exact correlation structure of k-satisfiability landscapes," in *Proceedings of GECCO*, 2009, pp. 365–372.

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

[17] "Software Product Line Online Tools(SPLOT)," 2013, http://www.splot-research.org/.

[18] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Using feature model knowledge to speed up the generation of covering arrays," in *VaMoS*, S. Gnesi, P. Collet, and K. Schmid, Eds. ACM, 2013, p. 16.