Title	Preliminary Study of Multi-objective Features Selection for Evolving Software Product Lines
Authors(s)	Brevet, David; Saber, Takfarinas; Botterweck, Goetz; Ventresque, Anthony
Publication date	2016-09-24
Conference details	18th International Symposium on Search Based Software Engineering (SSBSE 2016), Ralaigh, North Carolina, USA, 8-10 October 2016
Publisher	Springer
Item record/more information	http://hdl.handle.net/10197/7996
Publisher's statement	The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-47106-8_23.
Publisher's version (DOI)	10.1007/978-3-319-47106-8_23

Preliminary Study of Multi-objective Features Selection for Evolving Software Product Lines*

David Brevet[1,2], Takfarinas Saber[1], Goetz Botterweck[3], and Anthony Ventresque[1]

[1] Lero, School of Computer Science, University College Dublin, Ireland
takfarinas.saber@ucdconnect.ie, anthony.ventresque@ucd.ie
[2] University of Nantes, France, david.brevet1@etu.univ-nantes.fr
[3] Lero@UL, University of Limerick, Ireland, goetz.botterweck@lero.ie

Abstract. When dealing with software-intensive systems, it is often beneficial to consider families of similar systems together. A common task is then to identify the particular product that best fulfils a given set of desired product properties. *Software Product Lines Engineering* (SPLE) provides techniques to design, implement and evolve families of similar systems in a systematic fashion, with variability choices explicitly represented, e.g., as *Feature Models*. The problem of picking the 'best' product then becomes a question of optimising the *Feature Configuration*. When considering multiple properties at the same time, we have to deal with multi-objective optimisation, which is even more challenging.
While change and evolution of software systems is the common case, to the best of our knowledge there has been no evaluation of the problem of *multi-objective optimisation of evolving Software Product Lines*. In this paper we present a benchmark of large scale evolving Feature Models and we study the behaviour of the state-of-the-art algorithm (SATIBEA). In particular, we show that we can improve both the execution time and the quality of SATIBEA by feeding it with the previous configurations: our solution converges nearly 10 times faster and gets an 113% improvement after one generation of genetic algorithm.

Keywords: SPL, Multi-objective, Genetic Algorithm, Evolution

1 Introduction

Software Product Lines (SPL) is a branch of Software Engineering that aims at designing software products based on a composition of pre-defined software artefacts, increasing the reusability and personalisation of software products [5]. Software architects, when they design new products or adapt existing products, navigate a set of features in a Feature Model (FM). Each of these features represents an element of a software artefact that is of importance to some stakeholders. Through its structure and additional constraints, each FM describes all possible products as combinations of features. One of the issues with FMs is that they can be very large – for instance in our study we work with FMs composed of nearly 7,000 features and 350,000 constraints. *Optimising FMs*, i.e., *selecting the set of features* that could lead to potential real products, is

* This work was supported by Science Foundation Ireland grants 10/CE/I1855 and 13/RC/2094.

then a difficult problem [3]. This problem is also called SPL configuration as it consists in configuring products from the FMs. It is even more challenging as this problem is typically a multi-objective one: software designers and architects make their decisions based on various perspectives [4,8], such as, cost, technical feasibility or reliability.

Another related problem that has not been studied yet is the *feature selection in a multi-objective context when the FMs evolve*. It is not a surprise to say that software requirements and artefacts evolve constantly. For instance, stakeholders and customers often change their opinions about how applications should work, or new coding paradigms are introduced. FMs reflect that, and for instance, we have seen in our study that a large FM (such as the one behind the Linux kernel) evolves regularly and substantially (every few months a new FM is released with up to 7% difference from the previous one). In this context, it seems odd to generate random bootstrapping populations for the state-of-the-art genetic algorithms, such as, SATIBEA. It is tempting on the contrary to use the fact that FMs have evolved and that the SPL configurations generated previously, while not totally applicable, are close and can be adapted.

Our contributions in this paper are the following: (i) We propose a benchmark[4] for the analysis of evolving SPL; this data set has been generated following a study of the demographics and evolution of a large SPL (Linux kernel). This data set is important to provide a good evaluation of the different algorithms under different evolution scenarios; (ii) We propose *eSATIBEA* which is a modification of the state-of-the-art SATIBEA [4] for evolving SPL. eSATIBEA adapts previous solutions to new FMs to improve and speed-up the results of SATIBEA; (iii) We evaluate SATIBEA and eSATIBEA on the evolving SPL problem and show that eSATIBEA converges nearly 10 times faster and gets a 113% improvement after one generation of genetic algorithm.

Seeding is not a novel idea as such (e.g., see papers by Fraser and Arcuri [2] and Alshahwan and Harman [1]) – but usually seeding is done by taking a few good/previous solutions that are inserted in the initial population. In this paper we take all the previous solutions that we adapt to create a starting population. We also work on a large scale and very constrained search space, which is not always the case in models for which seeding is known to work.

The rest of this paper is organised as follows: Section 2 describes the problem of configuring evolving SPLs; Section 3 presents our benchmark; Section 4 evaluates SATIBEA and eSATIBEA; Section 5 concludes this paper.

2 Problem Definition

Feature Models can easily be represented by a set of features and relations (constraints) between them. Figure 1 shows a simple FM with 10 features linked by several relations, such as, 'alternative' between features 'Screen' : 'Basic', 'Colour' and 'High Resolution'. These relations define constraints: for instance, a 'Screen' can only be of one of 3 types 'Basic', 'Colour' or 'High Resolution'.

The objective of SPL engineering is to extract products from the FMs by selecting a subset of features $\mathcal{S} \subseteq \mathcal{F}$ which satisfies the FM $\mathcal{F}$ – and the requirements of the

[4] Available here: `http://hibernia.ucd.ie/EvolvingFMs/`.

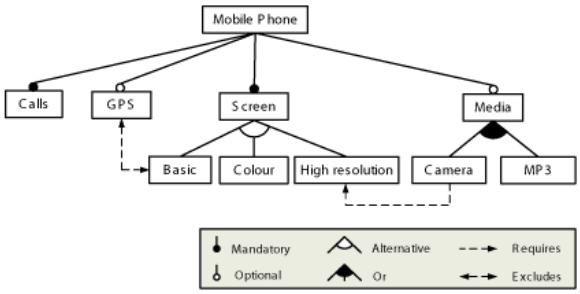

Fig. 1: Sample of a Feature Model

stakeholder/customer. Often, the SPL configuration problem is described as a satisfiability problem (SAT) [7], i.e., a problem where we try to find an assignment to variables (here, features) in the $\{True, False\}$ space. Let $f_i \in \{True, False\}$ be a decision variable set to '$True$' if the feature $F_i \in \mathcal{F}$ is selected to be part of $\mathcal{S}$ and '$False$' otherwise. An FM is equivalent to a conjunction of disjunctive clauses, forming a conjunctive normal form (CNF). Finding a product in the SPL is then equivalent to giving a value in $\{True, False\}$ to every variable/feature. For instance, in Figure 1 the FM would have the following clauses, among others: $(Basic \vee Colour \vee High\ resolution) \wedge (\neg Basic \vee \neg Colour) \wedge (\neg Basic \vee \neg High\ resolution) \wedge (\neg Colour \vee \neg High\ resolution)$, which describe the alternative between the three features. Now, software designers, when configuring a SPL, do not only look for possible products (satisfying the FM) but for products optimising some criteria – and there could be several of these criteria. This is why the problem of SPL configuration has been described as multi-objective. Here, following other classical approaches [4,8] we use 5 objectives: (i) number of selected features, (ii) number of selected features that were not used in the past, (iii) sum of known defects in the selected features, (iv) number of compatibility violations, and (v) cost of the selected features.

Evolution of the SPLs and FMs is known to be an important challenge for the domain, as they both represent long term investments [6]. For instance, in the next section we present a study of a large scale FM, the Linux kernel, and we show that every few months a new FM is released with up to 7% modifications among the features (features added or removed). The FM/SPL evolution perspective has not been addressed in the multi-objective feature selection literature – as far as we know. This is likely because the problem is a large and complex optimisation problem and the repairs/adaptation of previous solutions to new FMs unlikely to succeed. In this paper though we prove that it is not the case and that it is possible to feed solutions of previous FMs into new FMs, with good results. What we work with is a mapping between two FMs. Let us assume an FM FM_1 evolved into another FM FM_2. Some of the features $f_i^1 \in FM_1$ are mapped on to features $f_i^2 \in FM_2$ – they are the same or considered the same, while some of the features $f_i^1 \in FM_1$ are not mapped onto any features in FM_2 (f_i^1 has been removed) and features $f_i^2 \in FM_2$ have no corresponding features in FM_1 (f_i^2 has been added).

Obviously the same applied to constraints (removed from FM_1 or added to FM_2). The problem we address concerns adapting the solutions found previously for FM_1 to FM_2.

3 Towards a Benchmark for Feature-Model Evolution

We studied the largest open source FM we found: the Linux kernel [9] containing 6,888 features and 343,944 constraints (in its version 2.6.28). We evaluated the demographics (features, constraints) and evolution of 21 versions of the kernel: from version 2.6.12 to version 2.6.32. We observed that on average there was only 4.6% difference in terms of features between a version and the next: 21.22% of removed features and 78.78% of new (added) features. We also evaluated the size of the clauses/constraints in the problem, as we need to know how the constraints we add in the problem should look like.We found that a large proportion of the FMs' constraints have 6 features (39%), 5 features (16%), 18 features (14%) or 19 features (14%). From this study, we generated a synthetic benchmark of FM evolution based on the real evolution of the Linux kernel – hence a realistic benchmark but with more variability than in a real one, allowing us also to get several synthetic data sets for each evolution values. Our FM generator uses two parameters representing the percentage of feature modifications (added/removed) and the percentage of constraint modifications (added/removed). The higher those percentages are, the more different the new FM will be from the original one. Our FM generator uses the proportions we observed in the 20 FMs to generate new features/remove old ones, and to generate new constraints of a particular length. Values we use can be seen in our benchmark in Figure 2: from 5% of modified features and 1% of modified constraints (FM 5_1) to 20% of modified features and 10% of modified constraints (FM 20_10). In our evaluations (see next section) we generate 10 synthetic FMs for each values of the parameters.

4 Evaluation

This section evaluates two algorithms: SATIBEA, known in the literature as the best algorithm for multi-objective configuration of SPLs, and our contribution: eSATIBEA. We perform our evaluation on the benchmark described in the previous section and we compare the two algorithms using the hypervolume [11]. The hypervolume is a metric that indicates the space (in the n dimensions defined by the n objectives) dominated by the Pareto front of the solutions found by each algorithm. The bigger the hypervolume the better (i.e., more space is covered by the front of solutions).

The current state of the art in SPL configuration is the SATIBEA algorithm [4]. SATIBEA is the combination of a genetic algorithm (IBEA [10]) and a SAT solver. In particular, two steps are added to the genetic algorithm: 'smart mutation' and 'smart replacement', both applied with a certain probability, and both using the SAT solver to discover possible solutions to repair or replace infeasible solutions. We propose eSATIBEA, which aims at taking advantage of previous SPL configurations (when they exist) to feed in SATIBEA with previous solutions. Let's assume a FM FM_1 that evolves into another FM FM_2 over time (e.g., features and constraints added or removed). An SPL configuration was performed on FM_1 (e.g., using SATIBEA) and a set of solutions (S_1)

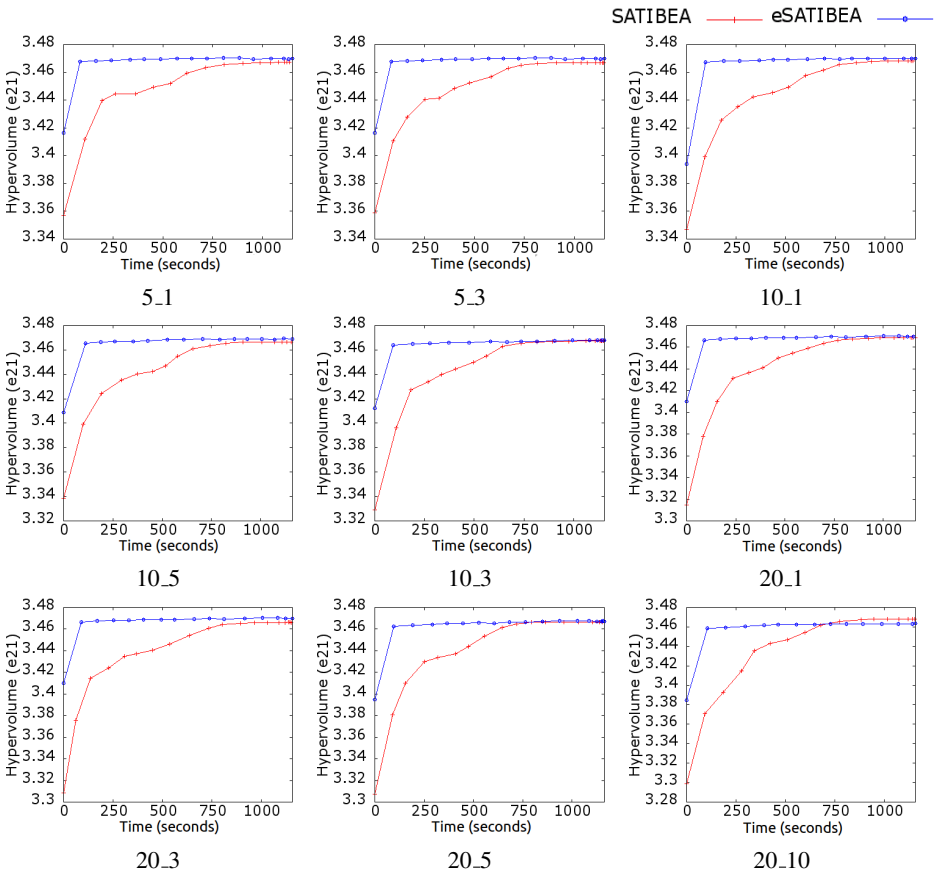

Fig. 2: Hypervolume of the solutions given by SATIBEA and eSATIBEA on evolved Linux kernels. We gave a label x_y to each evolution, 'x' representing the percentage of modified features and 'y' the percentage of modified constraints. We generated 10 evolved FMs for each combination and only show the average here.

was found. Now instead of randomly generating individuals for SATIBEA, we decide in eSATIBEA to adapt the set of solutions S_1 to the feature Model FM_2 and to give these solutions to SATIBEA. Our hope is obviously that these initial individuals will be of good quality, and anyway better than random solutions.

Figure 2 shows that both algorithms improve the hypervolume over time and eventually plateau after 1,200s. However, eSATIBEA takes advantage of the relatively good initial population and gets a better hypervolume for most data sets. Furthermore, we see that eSATIBEA converges quickly (i.e., less than 100s), whereas it takes SATIBEA more than 700s to reach a similar hypervolume than eSATIBEA. We also notice that eSATIBEA achieves an improvement of over 113% on average in comparison to SATIBEA at the end of the first generation of genetic algorithm (i.e., at 95s on average). This percentage decreases over time until 1% improvement on average. We see in Figure 2

(20_10) that SATIBEA gets an hypervolume slightly better than eSATIBEA by the end of the execution – while, as for other evolved FMs, eSATIBEA converges faster. This probably shows the limitation of our approach: 20_10 is a very different FM than the original one and SATIBEA, generating an initial population adapted to the new FM, does a better exploration of the space – while eSATIBEA stays close to a FM that is now obsolete.

5 Conclusion

This paper has presented a new problem: the configuration of Software Product Lines when the Feature Models they are based on evolves. To study this problem, we have proposed a benchmark using a survey of the evolution of a large (nearly 7,000 features and 350,000 constraints) FM. We have compared SATIBEA, the leading algorithm in the literature, and our contribution eSATIBEA (which takes an adaptation of the previous solutions as initial population) in this evolving context. We have shown that eSATIBEA outperforms SATIBEA, in particular it converges nearly 10 times faster and achieves an improvement of 113% after the first generation of genetic algorithm ($\simeq$ 100s).The two directions we plan to follow in the future are: an adaptation of the seed given to SATIBEA in eSATIBEA, and an improvement of eSATIBEA to overcome the problem of the plateau phase.

References

1. Nadia Alshahwan and Mark Harman. Automated web application testing using search based software engineering. In *ASE*, pages 3–12, 2011.
2. Gordon Fraser and Andrea Arcuri. The seed is strong: Seeding strategies in search-based software testing. In *ICST*, pages 121–130, 2012.
3. M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. Search based software engineering for software product line engineering: A survey and directions for future work. In *SPLC*, pages 5–18, 2014.
4. Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *ICSE*, pages 517–528, 2015.
5. Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In *FSE*, pages 70–84, 2014.
6. Andreas Pleuss, Goetz Botterweck, Deepak Dhungana, Andreas Polzer, and Stefan Kowalewski. Model-driven support for product line evolution on feature level. *JSS*, pages 2261–2274, 2012.
7. Richard Pohl, Kim Lauenroth, and Klaus Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *ASE*, pages 313–322, 2011.
8. Abdel Salam Sayyad, Joe Ingram, Tim Menzies, and Hany Ammar. Scalable product line configuration: A straw to break the camel's back. In *ASE*, pages 465–474, 2013.
9. Steven She. *Feature Model Synthesis*. PhD thesis, University of Waterloo, 2013.
10. E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *PPSN*, pages 832–842, 2004.
11. E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *PPSN*, pages 292–301, 1998.