Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines

Christopher Henard*, Mike Papadakis*, Mark Harman†, and Yves Le Traon*
*Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
†University College London, Gower Street, London
christopher.henard@uni.lu, michail.papadakis@uni.lu, mark.harman@ucl.ac.uk and yves.letraon@uni.lu

***Abstract*—Software Product Line (SPL) feature selection involves the optimization of multiple objectives in a large and highly constrained search space. We introduce SATIBEA, that augments multi-objective search-based optimization with constraint solving to address this problem, evaluating it on five large real-world SPLs, ranging from 1,244 to 6,888 features with respect to three different solution quality indicators and two diversity metrics. The results indicate that SATIBEA statistically significantly outperforms the current state-of-the-art ($p < 0.01$) for all five SPLs on all three quality indicators and with maximal effect size ($\hat{A}_{12} = 1.0$). We also present results that demonstrate the importance of combining constraint solving with search-based optimization and the significant improvement SATIBEA produces over pure constraint solving. Finally, we demonstrate the scalability of SATIBEA: within less than half an hour, it finds thousands of constraint-satisfying optimized software products, even for the largest SPL considered in the literature to date.**

I. Introduction

A Software Product Line (SPL) [1], [2], is a collection of related software products, all of which share some core functionality, yet each of which differs in some specific features. Software engineers use SPLs to increase software reusability and to rationalise software maintenance and evolution effort across a range of related products [3].

However, without automated support, this feature selection process is likely to be highly suboptimal: it requires the simultaneous satisfaction of multiple objectives, such as matching user preferences, minimizing product cost and satisfying technical feasibility constraints in feature spaces defined by many thousands of features. In such large, constrained spaces, human intuition is insufficient to find optimal or near optimal software products. In order to reduce configuration effort and optimize the resulting product choices, automated techniques for feature selection have been introduced. A recent survey of SPL product optimization can be found elsewhere [4].

The problem of feature selection was first addressed in 2008 by White et al. [5] who introduced an approach called Filtered Cartesian Flattening to select features from a feature model, but this was only able to cater for single optimization objectives. In 2011 Guo et al. [6], [7] introduced a genetic algorithm for the same problem, demonstrating that it outperformed Filtered Cartesian Flattening on synthetically generated SPLs, but did not present results for any real-world SPLs.

These previous approaches were all single objective approaches. Therefore they could not construct software products from SPLs for which multiple (perhaps conflicting and competing) objectives needed to be optimized. Sadly, such single objective solutions are unsuited to most real-world SPL feature selection problems (which are multi-objective). However, in 2011, Wu et al. [8] introduced a multi-objective optimization formulation that was evaluated on a Mail Server System case study.

In 2013 Sayyad et al. provided a detailed investigation of the multi-objective SPL feature selection problem in four related papers [9], [10], [11], [12] that collectively established the current state-of-the-art. Their first paper [11] demonstrated that search-based optimization can be used to find products that optimize multiple objectives. They evaluated on real-world SPLs, replicating their results [9], and reporting on parameter tuning effects [10]. Finally, Sayyad et al. introduced additional heuristics to improve the scalability of their approach [12], which is an important consideration for SPL optimization, since SPLs can be very large.

None of these previous approaches to SPL feature selection have included any explicit technique to handle constraints, leaving open the question of how best to optimize SPL feature selection in the presence of constraints. This is an important open question because most real-world SPLs are highly constrained [13], and solutions that fail to respect such constraints are likely to be rejected by both developers and their users.

Indeed, many constraint-violating solutions will prove to be simply *unbuildable*; constraints often determine whether or not a product can be feasibly constructed. Furthermore, this paper shows that concentrating on constraint-respecting solutions also allows the search to find software products that significantly outperform the state-of-the-art.

We introduce SATIBEA, a search-based SPL feature selection algorithm, augmented by constraint solving and two smart search operators. SATIBEA guides the automated search to constraint-respecting solutions that maximise multiple objectives in reasonable time. Our empirical study, which include the largest yet reported SPL, demonstrates that SATIBEA is a scalable and significant improvement over the current state-of-the-art.

The primary contributions of the paper can be summarised as follows:

1) We introduce SATIBEA, a new algorithm for SPL selection and evaluate it on 5 real-world SPLs, ranging from 1,244 to 6,888 features with respect to 3 quality indicators and two diversity measures. We perform 30 independent executions to support inferential statistical testing for significance and assessment of effect size.
2) We show that SATIBEA significantly outperforms the current state-of-the-art (with maximal effect size) according to all 3 solution quality indicators and for all 5 SPLs.
3) We demonstrate the importance of augmenting search with constraint solving in such constrained spaces as SPLs: We present results that show that our simple constraint solving approach alone can also significantly outperform the state-of-the-art with maximal effect size with respect to all 3 solution quality indicators in 3 SPLs including the largest one, Linux.
4) We demonstrate the added value of our combined approach with smart operators over constraint solving alone. Over the 15 comparisons (5 SPLs, each with 3 quality indicators) we find that SATIBEA significantly outperforms constraint solving alone in 13, and with maximal effect size in 11.
5) We demonstrate SATIBEA's scalability. Scalability is a known and important issue for both SPLs [12], [13], [14] and search-based software engineering [15].

The remainder of the paper is organized as follows: Sections II and III introduce and motivate the concepts and the propositions underlying the present paper. Section IV details the proposed approaches. The studied research questions and the experimental setup are detailed in Sections V and VI. Experimental results are presented and discussed in Sections VII and VIII. Finally, Section IX examines work related to the present one and Section X concludes the paper.

II. Background

This section introduces background concepts on SPLs and multi-objective optimization that are used in the paper.

A. Software Product Line Engineering

Software engineers build many variations of their systems in order to match the specific needs of particular clients [16]. Software Product Line Engineering (SPLE) is a software development paradigm designed to handle this situation. It involves the creation and the management of an SPL which encompasses the different variants, called products. SPLE appeared in 1990 with the development of Feature-Oriented Domain Analysis [17]. The benefits of SPLE include the reduction of the maintenance effort, lower development costs and a faster time to market [3]. The variabilities and commonalities among software products are expressed in terms of features [18]. Each feature is an abstraction of a functionality or property of the software products.

1) Feature Model: Feature Models (FMs) are the standard and compact representation of the possible products of an SPL [19], [20]. An FM defines the valid feature combinations [18] by expressing constraints between them. As an example, consider the FM depicted in Figure 1. It contains 9 features. Some features are mandatory (included in every product), e.g., the "draw" feature. Other features are constrained to co-occur. For instance, the "color" feature requires the "color palette".

An FM can be translated to a Boolean formula in Conjunctive Normal Form (CNF). Such formulas are a conjunction of n clauses $c_1, ..., c_n$, where a clause is a disjunction of m literals. A clause is a constraint between some features of the FM and a literal is a feature that is selected (f_j) or not ($\overline{f_j}$): $FM = \bigwedge_{i=1}^{n}(\bigvee_{j\in[1;m]} l_j)$, where $l_j = f_j$ or $\overline{f_j}$. For instance, the FM of Figure 1 contains $m = 9$ features and encompasses $n = 18$ constraints represented as follows in CNF: $f_1, (\overline{f_2} \vee f_1), (\overline{f_1} \vee f_2), (\overline{f_3} \vee f_1), (\overline{f_1} \vee f_3), (\overline{f_4} \vee f_1), (\overline{f_5} \vee f_1), (\overline{f_1} \vee f_5), (\overline{f_6} \vee f_3), (\overline{f_7} \vee f_3), (\overline{f_3} \vee f_6 \vee f_7), (\overline{f_8} \vee f_5), (\overline{f_9} \vee f_5), (\overline{f_5} \vee f_8 \vee f_9), (\overline{f_8} \vee \overline{f_9}), (\overline{f_7} \vee f_4), (\overline{f_4} \vee \overline{f_8}), (\overline{f_9} \vee f_4)$. The corresponding CNF formula is a conjunction of all the constraints: $FM = f_1 \wedge (\overline{f_2} \vee f_1) \wedge (\overline{f_1} \vee f_2) \wedge (\overline{f_3} \vee f_1) \wedge (\overline{f_1} \vee f_3) \wedge (\overline{f_4} \vee f_1) \wedge (\overline{f_5} \vee f_1) \wedge (\overline{f_1} \vee f_5) \wedge (\overline{f_6} \vee f_3) \wedge (\overline{f_7} \vee f_3) \wedge (\overline{f_3} \vee f_6 \vee f_7) \wedge (\overline{f_8} \vee f_5) \wedge (\overline{f_9} \vee f_5) \wedge (\overline{f_5} \vee f_8 \vee f_9) \wedge (\overline{f_8} \vee \overline{f_9}) \wedge (\overline{f_7} \vee f_4) \wedge (\overline{f_4} \vee \overline{f_8}) \wedge (\overline{f_9} \vee f_4)$.

2) Product Configuration: A product configuration C corresponds to the features that are present or not in a given product: $C = \{l_1, ..., l_m\}$ where $l_j = f_j$ or $\overline{f_j}$. For instance, with respect to Figure 1, $C_1 = \{f_1, f_2, f_3, f_4, f_5, \overline{f_6}, f_7, \overline{f_8}, f_9\}$ is a configuration representing the software product proposing all the features except rectangular selection and black and white rendering. This configuration is valid, since it satisfies the constraints of the FM described in the previous subsection. By contrast, $C_2 = \{f_1, \overline{f_2}, f_3, f_4, f_5, \overline{f_6}, f_7, \overline{f_8}, f_9\}$ violates the constraint $(\overline{f_1} \vee f_2)$ and is thus invalid.

B. Multi-objective Optimization

Multi-Objective Optimization (MOO) refers to the process of optimizing more than one objective at the same time. The aim of these approaches is to search for optimal (or nearly optimal) solutions requiring trade-offs between two or more conflicting objectives. Let X be the set of all the possible product configurations of an SPL and let $v = [F_1(x), ..., F_k(x)]^T$ be a vector of k objective functions. If each objective has to be

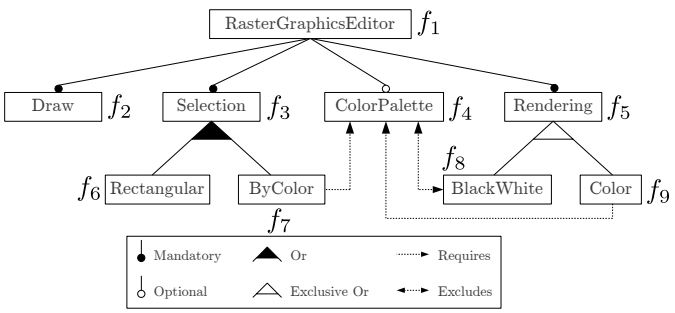


Fig. 1: A feature model of a raster graphics editor. It represents the 9 features of the software product line and their constraints.

minimized, MOO aims at finding $x_1, ..., x_k$, i.e., the solutions to the problem, such as v is minimized. The minimization of v is the process of optimizing simultaneously the k objective functions [21].

Let x_1 and x_2 be two potential solutions to the problem. We say that x_1 dominates x_2, written as $x_1 \succ x_2$ if and only if $\forall i \in \{1, ..., k\}\, F_i(x_1) \leq F_i(x_2)$ and $\exists i \in \{1, ..., k\}\, F_i(x_1) < F_i(x_2)$. Given $x_1, ..., x_n$ potential solutions to the MOO problem, the Pareto front corresponds to the subset of these potential solutions that are non-dominated by the others. An example of Pareto front is illustrated by Figure 2 for two objectives F_1 and F_2 to minimize. In this example, x_1, x_2, x_4, x_6 and x_7 are in the Pareto front set since they are not dominated by any other solution. By contrast, x_{10} is dominated (among others) by x_2 ($x_2 \succ x_{10}$). So, it does not lie on the front. Finally, we denote as Pareto front size the number of solutions in the Pareto front.

C. State-of-the-art, the IBEA method

The Indicator-Based Evolutionary Algorithm (IBEA) [22] is an evolutionary MOO technique using quality indicators to guide the search towards the optimal solutions. IBEA has the ability to exploit user preferences. The advantages of this algorithm over other search techniques for the SPL configuration problem have been shown in [11], [12].

Sayyad *et al.* [11] proposed setting the number of constraints that are violated as a minimization objective within the search process in order to deal with the SPL constraints. The user preferences are also modeled as additional optimization objectives. The approach applies the standard mutation, i.e., flipping bits of the offspring with a specific probability, and crossover operators. Their results provide evidence that this practice can lead to invalid and marginally invalid configurations. They also suggest that IBEA is capable of providing a wide range of valid configurations that exploit and optimize user preferences.

The results of Sayyad *et al.* were reinforced by the study of Olaechea *et al.* [23], [24] who demonstrated, on small models, that IBEA is capable of finding the optimal solutions. Olaechea *et al.* also showed that is feasible to compute exact solutions when considering models with fewer than 45 features. This bound indicates the need for approximation algorithms, such as IBEA, for the cases of larger models.

Empirical evidence has been provided to show that by enhancing the initial population of the algorithm with one valid configuration, called *seed*, IBEA is capable of scaling on very large FMs has also been provided [12]. According to the studies of Sayyad *et al.*, one seed that is rich, i.e., one configuration with many features selected, is adequate for improving the search process and more effective than using many seeds. We only consider large SPLs and thus, we compare with this approach which forms the current state-of-the-art. In the rest of the paper we refer to it as the *state-of-the-art* or as the *IBEA* approach.

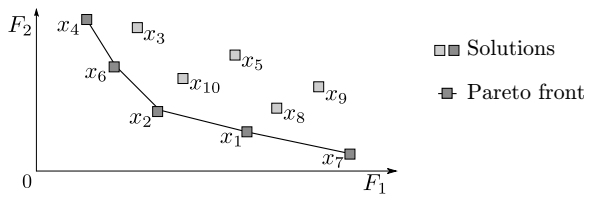


Fig. 2: An example of Pareto front with two objectives F_1 and F_2 to minimize. The solutions x_1, x_2, x_4, x_6 and x_7 are in the Pareto front since they are not dominated by any other.

III. Motivation

Automatic configuration of SPLs is a challenging problem. The problem is very hard for large models where most of the existing approaches fail. One such approach is the random one. To demonstrate the difficulty of the problem, we implement a baseline approach that randomly selects and deselects features of the model. By applying the baseline for 30 minutes we expect to generate a large set of configurations some of which, we might suppose, would turn out to be valid according to the constraints. However, after the application on our studied models (refer to Section VI-A for further details), we found *no* valid configuration out of the 57,303,105 solutions produced for the 5 subjects in total.

This simple fact shows that the problem is hard and cannot be tackled with ad-hoc solutions. Additionally, the results of Sayyad *et al.* [12] show that both the IBEA and NSGA-II approaches fail to provide any valid solution after 30 minutes of execution when their population is not seeded. Similarly, the study of Olaechea *et al.* [23], [24] demonstrates that it is infeasible to compute exact solutions for models with more than 45 features. Collectively, these results highlight the challenges posed by constrained industrial scale SPLs.

IV. The proposed approaches

One of the most challenging SPL optimization tasks is the automatic generation of valid configurations. The current state-of-the-art uses IBEA to search and find valid solutions. An alternative to search would involve the use of a SAT solver [14]. In this case, a valid configuration is a satisfiable model found by the solver. To this end, one might attempt to enumerate all the valid solutions of a model and select those that are optimal with respect to the other objectives. However, the large number of valid configurations makes this simplistic approach infeasible [24]. As a result, some form of search-based technique is needed.

To effectively perform search, we seek to combine the benefits of both constraint solving and searching in a complimentary way. The question this raises is how best to perform such a combination. To achieve this, two key aspects are considered: diversity promotion and search using smart operators. These aspects are taken into account in our approach called *SATIBEA*. We also define a "filtered" technique which only bestows the diversity promotion. Doing so allows to empirically assess its contribution to SATIBEA success in isolation.

A. Diversity Promotion

We wish to promote maximal diversity of SAT solutions in a cheap way. We do this by randomly permuting the parameters that control the search for constraint-satisfying solutions processed by the SAT solver. More specifically, there are three different SAT parameters that we permute:

1) **Constraint order**. This is the order in which the constraints are considered.
2) **Literal order**. This is the order in which the literals of each constraint are ordered.
3) **Phase selection**. This is the order $\{true, false\}$ in which assignments to variables are instantiated.

By randomly permuting these three parameters at each iteration of the SAT execution, we increase the diversity of solutions found. To empirically assess the degree of Diversity Promotion (DP) this creates, we use a dissimilarity metric, as it is defined in [14]. Based on the Jaccard distance, this metric captures degrees of difference between the selected and unselected features of two configurations. The metric takes values between 0 and 1. A value of 1 signifies that the two configurations differ completely, while, 0 signifies that the two considered configuration are the same.

Table I records the results of the above-mentioned dissimilarity metric for solutions produced, on a set of subject models, with and without DP. These subjects are introduced in Section VI-A. Specifically, Table I records a) the average dissimilarity between two configurations that are consecutively generated by calling the solver 1,000 times, b) the set dissimilarity, i.e., the dissimilarity between any two configurations from a set of 1,000 configurations produced by the solver and c) the percentage increase in the diversity of the configurations as measured by the a) and b) cases. The dissimilarity between two consecutive configurations C_i, C_j is measured by $d(C_i, C_j) = \frac{|C_i \cup C_j| - |C_i \cap C_j|}{|C_i \cup C_j|}$. The dissimilarity of a set of n configurations is measured by $D(C_1, ..., C_n) = \frac{1}{\binom{n}{2}} \sum_{j>i}^{n} d(C_i, C_j)$. Finally, the increase in the diversity is calculated as follows: $\text{increase} = \frac{(\text{with DP} - \text{without DP})}{\text{without DP}} \times 100$. As it can be seen in Table I, the permutation of the SAT parameters allows the diversity of the solutions to increase by 2,768% in the worst case for the consecutive calls to the solver. For the set of configurations, the diversity increase was 161% in the worst case and more than 39,000% in the best case.

TABLE I: Dissimilarity with and without Diversity Promotion (DP) on 1,000 configurations per model.

	Consecutive configurations			**Set of configurations**		
	without DP	with DP	increase	without DP	with DP	increase
Linux	0.0004	0.5934	148,250%	0.0015	0.5937	39,487%
uClinux	0.0036	0.2807	7,697%	0.1080	0.2814	161%
Fiasco	0.0066	0.1892	2,768%	0.0436	0.1869	329%
FreeBSD	0.0022	0.5891	26,677%	0.0074	0.5897	7,991%
eCos	0.0046	0.5429	11,702%	0.0304	0.5426	1,685%

B. "Smart" Operators

We introduce two operators that are "smart" in the sense that they are constraint-aware and using diversity promotion.

1) Smart Mutation: This mutation operates by finding the features that are not involved in the violations of constraints. It keeps their values and asks the solver to find a solution for the rest by assuming the values of the rest of the features. Consider an FM with 5 features and 3 constraints: $FM = (\overline{f_1} \vee f_5) \wedge (\overline{f_2} \vee f_3) \wedge (f_2 \vee f_5)$. The configuration $C = \{\overline{f_1}, \overline{f_2}, f_3, f_4, \overline{f_5}\}$ is invalid because the two constraints $(f_1 \vee f_5)$ and $(f_2 \vee f_5)$, which involve the features f_1, f_2 and f_5, are violated. We remove the assignment of these features and make C partially valid, i.e., $C : C_{\text{partial}} = \{\_, \_, f_3, f_4, \_\}$. This partial configuration is given to the SAT solver, which will complete it, to return a valid configuration. For instance, it can return the following configuration $C' = \{f_1, f_2, f_3, f_4, \overline{f_5}\}$. As a result, C has been mutated into C'.

2) Smart Replacement: This operator randomly picks a configuration from the solutions and replaces it with a new valid one, improving the quality and diversity of the solutions.

C. The SATIBEA approach

SATIBEA augments IBEA [22] with the smart operators. Diversity promotion is used in the optimization process through these two operators. SATIBEA also employs a form of memory by keeping track of all the valid configurations produced by the algorithm. Based on these solutions, we compute the Pareto front. Thus, the population is evolved via the four following operators:

1) **Mutation**. This is the standard bit-flip operator of IBEA. It iterates over the bits, i.e., the feature options, of the offspring, i.e., the configuration, and flips them with a specific probability.
2) **Crossover**. This is the "standard" single-point crossover operator of IBEA. It combines two solutions, i.e., configurations, by replacing the bits of the first one, from the beginning of the offspring up to the crossover point, with those of the second one.
3) **Smart Mutation**, as described in Section IV-B1.
4) **Smart Replacement**, as introduced in Section IV-B2.

D. The Filtered approach

To investigate the contribution of the diversity promotion to SATIBEA's performance, we also define a simple algorithm that simply randomly samples over diversity promoted SAT solutions. We refer to this approach as the *Filtered* one.

V. Research Questions

We first empirically evaluate SATIBEA against the current state-of-the-art [11], [12]. This is a natural first research question, since there is no point in evaluating further if our new algorithm cannot convincingly outperform the state-of-the-art.

RQ1. How does the SATIBEA compare with the current state-of-the-art?

Since the results of RQ1 indicate that SATIBEA does, indeed, convincingly outperform the state-of-the-art, we turn to the question of examining why. Naturally, since one of our

primary novelties lies in the incorporation of SAT solving into the search for constraint-respecting solutions, we next investigate and report on the effectiveness of SAT solving alone. How well would SAT solving perform against the current state-of-the-art on its own? This motivates RQ2:

RQ2. How well does the state-of-the-art perform against constraint solving alone (randomly selected solutions filtered by SAT, i.e., the Filtered approach)?

Perhaps surprisingly, we found that the Filtered approach outperforms the state-of-the-art. This provides compelling evidence that constraint solving does have an important role to play in the search for optimized products, automatically configured from SPLs. However, it also raises a further question: does SATIBEA significantly outperform constraint solving alone? If the answer is 'no', then all the value in our new SATIBEA approach derives from our incorporation of constraint solving, with search-based optimization and our smart mutation operators offering little added value. In order to check that this is *not* the case, we investigate RQ3 below:

RQ3. How well does SATIBEA perform against constraint solving alone (randomly selected solutions filtered by SAT, i.e., the Filtered approach)?

At this point in our study we will have considered whether our new algorithm SATIBEA outperforms the state-of-the-art (RQ1), whether constraint solving plays an important role in its performance (RQ2) and whether SATIBEA adds value to the search for constraint-respecting optimized software products over-and-above pure constraint solving alone (RQ3). Our final question concerns the execution time required to achieve these results. Even if SATIBEA convincingly outperforms all alternatives, this will be of little consequence if it does not scale well to the challenges of very large SPLs involving billions of possible configurations over thousands of features. We therefore conclude our study by reporting on the time taken to complete the execution of SATIBEA on the largest SPL for which results have been reported in the literature to date.

RQ4. What is the execution time required to find constraint-respecting optimized software products from the largest SPL hitherto considered in the literature?

VI. Experimental Setup

This section presents the settings of the conducted experiments. Specifically, it describes the subjects, the optimization objectives and the employed metrics.

A. Subjects

The study uses 5 FMs taken from the Linux Variability Analysis Tools (LVAT) repository[1]. The characteristics of the FMs are described in Table II. For each of them, it presents the version used, and the number of features and constraints it contains. Following the evaluation approach used by Sayyad *et al.* [11], [12], each feature of each FM has been augmented with 3 attributes: *cost*, *used before* and *defects*. The values for these attributes have been set arbitrarily with a uniform distribution: *cost* takes real values between 5.0 and 15.0, *used before* takes Boolean values and *defects* takes integer values between 0 and 10. The following dependency among these attributes is used: if (not *used before*) then *defects* = 0.

[1] http://code.google.com/p/linux-variability-analysis-tools

TABLE II: Feature models used in the empirical study.

Feature model	Version	Features *(mandatory)*	Constraints
Linux [25]	2.6.28.6	6,888 *(58)*	343,944
uClinux [26]	20100825	1,850 *(7)*	2,468
Fiasco [26]	2011081207	1,638 *(49)*	5,228
FreeBSD [25]	8.0.0	1,396 *(3)*	62,183
eCos [25], [27]	3.0	1,244 *(0)*	3,146

B. Optimization Objectives

In this study, we are measuring the following 5 objectives:

1) *Correctness*. We seek to minimize the constraints of the FM that are violated by a configuration.
2) *Richness of features*. We seek to minimize the number of deselected features in a configuration.
3) *Features that were used before*. We seek to minimize the features that were not used before, i.e., minimize the number of "false" for this attribute.
4) *Known defects*. We seek to minimize the number of known defects in a configuration.
5) *Cost*. We seek to minimize the cost of a configuration.

In practice, based on the needs and the historical data of engineers, other objectives can be also used. We selected these five objectives to ensure identical settings as those reported for the state-of-the-art [12].

C. Settings

All the experiments were performed on a Quad Core@2.40 GHz with 24GB of RAM. To enable a fair comparison with the state-of-the-art, we used exactly the same settings as the ones of Sayyad *et al.* [12]. These settings are: population size *300*, archive size *300*, crossover rate *0.05* and *0.001* mutation probability. The mutation probability refers to the probability that an optional feature of the model will be flipped. Regarding the configurations, we systematically set mandatory features (features that have to be present in any configuration) as selected and dead ones (features that cannot be part of any configuration) as unselected in the initial population of IBEA. We also prevented IBEA from flipping these features during the mutation process. Flipping these features always leads to invalid configurations. Thus, this practice helps IBEA to find valid configurations. The same evaluations settings were undertaken in the study of Sayyad *et al.* [12].

We carefully followed all the recommendations of Sayyad *et al.* in our experiments. Unfortunately, it is impossible to produce the same seeds. Since the work of Sayyad *et al.* [12] is not currently accompanied by any data or implementation, we simply followed the guidelines they give in their paper. Therefore, we have produced seeds using the solver by maximizing the number of selected features. This is done by setting the SAT parameter "phase selection" to assign $true$ to the

TABLE III: Selected features in the seeds used by IBEA.

Feature Model	Selected features	Selected features in [12]
Linux	6,265	5,704
uClinux	613	455
Fiasco	338	575
FreeBSD	1,088	946
eCos	1,148	967

variables. Note that this parameter was also used for diversity promotion (see Section IV-A). We thus produce one "rich" seed per model, as suggested by Sayyad *et al.* [12]. Table III describes the number of features selected in each seed.

Similarly, the settings for SATIBEA are the same as for IBEA, i.e., population size *300*, archive size *300*, crossover rate *0.05*. The probability to use the standard mutation of IBEA (bitflip), which mutates a chromosome is set to 0.98. The probability to flip a feature is set to *0.001* per feature. The probabilities of mutating using the smart mutation and the smart replacement is 0.01 for both cases. We employed the Sat4j SAT solver [28] and used the jMetal framework [29] for the implementation of IBEA and for the quality and diversity metrics (see Section VI-D). We independently applied each approach 30 times per FM with 30 minutes of execution time for each algorithm. Invalid configurations were discarded for all the studied techniques. Recall that invalid configurations are useless in practice.

D. Metrics

To evaluate the studied approaches, we follow two directions: 1) we measure the proximity of the solutions found from the optimal ones, i.e., their quality, and 2) we evaluate the diversity of the solutions. Note that diversity is only useful when there is quality: a single diamond is preferable to an arbitrary number of diverse glass fragments. In other words, it is useless to have diverse solutions that are all dominated by a single one. Therefore, the diversity metrics should be considered only when comparing solutions of similar quality.

Since the global optimum cannot be known in all cases (as with all NP-hard problems), a reference front is used in evaluation. It consists of the best solutions found by all the studied approaches and it is defined as follows: Given n Pareto fronts $A_1, ..., A_n$, and if $1 \leq j \leq m \leq n$, the reference front A_{ref} is defined as: $A_{\text{ref}} = \{x_1, ..., x_m \,|\, (\forall x_j \in A_{\text{ref}})(\nexists x' \in \bigcup_{i=1}^{n} A_i)(x' \succ x_j)\}$. It should be noted that $A_{\text{ref}} \subseteq A_i \cup ... \cup A_n$.

1) Quality Metrics: These metrics ensure that we find high quality solutions. Following the evaluation approach suggested by Knowles *et al.* [30], we use three metrics to evaluate the quality of the configurations of the Pareto front: *Hypervolume*, *Epsilon* and *Inverted Generational Distance*.

a) Hypervolume (HV): This metric represents the volume of the objective space that is dominated by the Pareto front A. It evaluates how well a Pareto front fulfills the optimization objectives. It is written HV and defined in [31] as follows: $\text{HV}(A) = \lambda\left(\bigcup_{x \in A}[F_1(x), r_1] \times \cdots \times [F_k(x), r_k]\right)$, where $\lambda(S)$ is the Lebesgue measure [32] of a set S, k is the number of objectives, $r = [r_1, ..., r_k]$ is the reference point and $[F_1(x), r_1] \times \cdots \times [F_k(x), r_k]$ is the k-dimensional hypercuboid consisting of all the points dominated by the point x. The reference point is the maximum value that belongs to the reference front. A higher HV denotes a better Pareto front.

b) Epsilon (ϵ): This metric measures the shortest distance that is required to transform every solution in a Pareto front A to dominate the reference front [30]. If $x = [x^1, ..., x^k]^T \in \mathbb{R}_k^+$ is a solution, it is defined as [33]: $\epsilon(A, A_{\text{ref}}) = \inf_{x \in \mathbb{R}} \{\forall x' \in A_{\text{ref}} \exists x \in A \,|\, x \succeq_\epsilon x'\}$, where $x \succeq_\epsilon x'$ if and only if $\forall 1 \leq i \leq k : x^i \leq \epsilon \cdot x'^i$. This indicator denotes how close A is to the reference front and thus, lower values are preferable.

c) Inverted Generational Distance (IGD): This metric is the average distance from the solutions belonging to the reference front to the closest solution in a Pareto front A [34]. IGD is defined as follows: $\text{IGD}(A, A_{\text{ref}}) = \frac{\sum_{x' \in A_{\text{ref}}} d(x', A)}{\text{PFS}(A_{\text{ref}})}$, where $d(x', A)$ is the minimum Euclidean distance between x' and the other points in A and PFS is the Pareto Front Size (see Section VI-D2a). For ϵ, the lower the value of IGD, the closer A is to the reference Pareto front.

2) Diversity Metrics: These metrics ensure that the decision maker has a variety of solutions to choose. We use two diversity metrics: the *Pareto front size* and the *Spread* of the solutions in the explored space.

a) Pareto Front Size (PFS): This metric is the number of solutions in a Pareto front A. It is calculated as the cardinality of the Pareto front set, i.e., $\text{PFS}(A) = |A|$. A higher Pareto front size is preferred since more options are given to the user. However, this is only important when high quality is preserved.

b) Spread (S): The spread measure defines the extent of spread in the solutions of the Pareto front A. It is defined in [35] as follows: $\text{S}(A) = \frac{d_f + d_l + \sum_{i=1}^{\text{PFS}(A)-1} |d_i - \overline{d}|}{d_f + d_l + (\text{PFS}(A)-1)\overline{d}}$, where d_i is the Euclidean distance between consecutive solutions of A, $\overline{d}$ is the average of the d_i's and d_f and d_l are the Euclidean distance between the extreme solutions and the boundary solutions of A. A higher spread denotes a better Pareto front since it reflects more diverse solutions, i.e., distributed among all the optimization objectives.

E. Statistical Analysis and Tests

To check the statistical significance of the differences between the algorithms, we performed a statistical test using the MannWhitney U test (two-tailed) at a 5% significance level. It is a non-parametric test and thus, it makes fewer assumptions regarding the underlying populations. Based on this test, we obtain an estimation about the probability, i.e., p-value, that one algorithm gives different values than the other. Furthermore, to reduce the threats of having type I errors in the cases of multiple comparisons, i.e., incorrect rejection of a true null hypothesis, we also consider the standard Bonferroni adjustment [36]. This is a conservative but safe adjustment because it reduces the chances of type I errors. Following the advice of Arcuri and Briand and Wohlin *et al.* [36],

TABLE IV: State-of-the-art VS the proposed approaches: comparison in terms of quality metrics, i.e., hypervolume (**HV**), epsilon (ϵ) and inverted generational distance (**IGD**), and diversity metrics, i.e., Pareto front size (**PFS**), spread (**S**) on 30 independent runs per approach. Higher values are preferred for **HV**, **PFS** and **S**. Lower values are preferred for ϵ and **IGD**.

			IBEA (I)		**Filtered (F)**		**SATIBEA (SI)**		**SI VS I**		**F VS I**		**SI VS F**	
			median	avg	median	avg	median	avg	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$
Linux	Quality	**HV**	7.75E-7	1.00E-6	0.1133	0.119	0.1624	0.1627	3.02E-11	1	3.02E-11	1	4.62E-10	0.97
		ϵ	0.9084	0.9082	0.1623	0.1616	0.0733	0.0721	2.96E-11	1	2.97E-11	1	3.02E-11	1
		IGD	0.0177	0.0177	0.0014	0.0014	0.0003	0.0003	3.02E-11	1	3.02E-11	1	3.02E-11	1
	Diversity	**PFS**	7	7.433	117	116.7	111.5	110.8	2.54E-11	1	2.56E-11	1	4.43E-07	0.11
		S	0.9988	0.9988	0.8679	0.8844	0.9004	0.8997	1.11E-06	0.14	6.77E-11	0.21	0.40	0.56
uClinux	Quality	**HV**	0.1956	0.1959	0.1300	0.1297	0.3030	0.3032	3.02E-11	1	3.02E-11	0	3.02E-11	1
		ϵ	0.1029	0.1018	0.0783	0.0783	0.0107	0.0101	2.87E-11	1	2.38E-11	1	2.30E-11	1
		IGD	0.0013	0.0013	0.0016	0.0016	0.0001	0.0001	2.97E-11	1	2.60E-10	1	2.97E-13	1
	Diversity	**PFS**	106	106.9	982.5	981.9	2,934	2,941	2.95E-11	1	2.95E-11	1	3.02E-11	1
		S	0.5809	0.5804	0.5125	0.5138	0.3610	0.3574	3.02E-11	0	1.20E-08	0.07	3.02E-11	0
Fiasco	Quality	**HV**	0.0238	0.0226	0.2879	0.2877	0.2894	0.2897	3.02E-11	1	3.02E-11	1	2.92E-09	0.95
		ϵ	0.0833	0.0842	0.0036	0.0036	0.0036	0.0035	1.01E-11	1	1.10E-11	1	0.67	0.45
		IGD	0.0064	0.0065	0.0002	0.0002	0.0002	0.0002	3.02E-11	1	3.02E-11	1	0.49	0.56
	Diversity	**PFS**	10	9.933	2,232	2,231	1,928	1,920	2.74E-11	1	2.74E-11	1	3.01E-11	0
		S	0.9721	0.9282	0.3039	0.3037	0.2959	0.2953	3.02E-11	0	3.02E-11	0	4.08E-05	0.20
FreeBSD	Quality	**HV**	0	0.0257	0.1323	0.1328	0.2485	0.2488	8.38E-10	1	8.39E-10	0.95	3.02E-11	1
		ϵ	1	0.7926	0.1861	0.1860	0.0953	0.0953	1.61E-11	1	1.61E-11	1	2.98E-11	1
		IGD	1	0.6022	0.0013	0.0013	0.0002	0.0002	1.61E-11	1	1.60E-11	1	3.02E-11	1
	Diversity	**PFS**	0	5.333	476.5	475.7	1,386	1,383	1.62E-11	1	1.60E-11	1	3.00E-11	1
		S	0	0.3990	0.5959	0.5949	0.7197	0.7185	3.02E-11	0.62	1.60E-11	0.62	3.02E-11	1
eCos	Quality	**HV**	0.0399	0.0462	0.2591	0.2591	0.2876	0.2876	3.02E-11	1	3.02E-11	1	3.02E-11	1
		ϵ	0.5974	0.5906	0.0975	0.0975	0.0382	0.0386	3.02E-11	1	2.53E-11	1	2.53E-11	1
		IGD	0.0013	0.0013	0.0002	0.0002	5.80E-05	5.77E-05	3.02E-11	1	3.02E-11	1	3.02E-11	1
	Diversity	**PFS**	50	55.17	2,886	2,881	14,421	14,064	3.00E-11	1	2.99E-11	1	3.02E-11	1
		S	0.9386	0.9414	0.4551	0.4548	0.5368	0.5364	3.02E-11	0	3.02E-11	0	3.02E-11	1

[37], we also report the non-parametric effect size measure, $\hat{A}_{12}$, introduced by Vargha and Delaney [38]. It measures the extent to which the first algorithm outperforms the second one. According to Vargha and Delaney [38], the differences between populations are considered as small, medium and large when $\hat{A}_{12}$ is over 0.56, 0.64, and 0.71, respectively.

VII. Experimental Results

The results for each approach are analyzed in Section VII-A. Sections VII-B and VII-C discuss the RQ1-RQ3. Finally, Section VII-D presents results regarding the execution time of SATIBEA on the largest SPL of the literature.

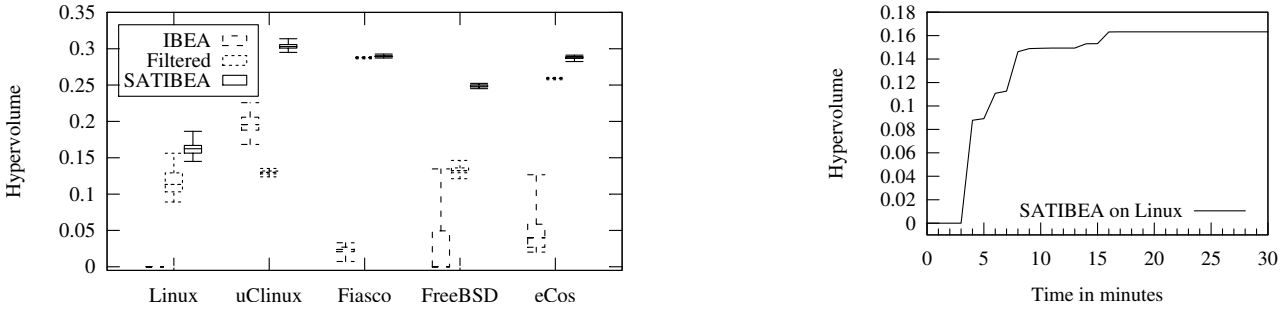
A. Results

This section presents the result of the approaches when applied to the five models. These results are recorded in Table IV. This table is composed of two parts. The columns *IBEA (I)*, *Filtered (F)* and *SATIBEA (SI)*, records the measured details about each approach. In particular, it records, for 30 executions the median and average (column avg) values of the measured metrics. The second part, i.e., columns *SI VS I*, *F VS I*, and *SI VS F*, records the results of the statistical analysis results, i.e., the p-values and the effect sizes $\hat{A}_{12}$. The rows of the table record the results per examined model, hypervolume measure (rows HV), Epsilon (rows ϵ), Inverted Generational Distance (rows IGD), Pareto front size (rows PFS) and spread metric (rows S) for the the 30 runs per approach. In addition, Figure 3a shows the distribution of the HVs on the 30 runs for all the models and the evolution of the HV over time for Linux is depicted in Figure 3b.

B. Comparing SATIBEA and Filtered with the state-of-the art, answering RQ1 and RQ2

Our results indicate that SATIBEA outperforms the current state-of-the-art (IBEA). The statistical analysis results, with respect to the quality metrics, column *SI VS I*, suggest that all the differences are significant with maximal effect size ($\hat{A}_{12} = 1.0$). The resulting p-value are so small that nothing changes when applying the Bonferroni correction. Regarding the diversity metrics, SATIBEA provides better results with respect to PFS but worse with respect to S. However, S is a diversity metric which is only important when there is quality in the solutions found, e.g., HV.

Additionally, these results reveal that SATIBEA is much



(a) Distribution of the hypervolumes on the 30 runs

(b) Hypervolume over time for SATIBEA on Linux

Fig. 3: Distribution and evolution of the hypervolumes.

better when it is applied on the two most heavily constraint models, i.e., Linux and FreeBSD. In the median case of Linux (Figure 3), SATIBEA produces configurations that cover approximately 209,548 times more wider space, i.e., hypervolume, than IBEA. In the median case of FreeBSD, IBEA failed to find even one solution that is valid. Furthermore, the solutions found by SATIBEA do not only provide more options, as shown by the PFS values, than the state-of-the-art but they are also more stable, as shown in Figure 3a. All these results indicate the superiority of our method.

Our results also indicate that the Filtered approach is better than the state-of-the-art. The statistical analysis results, with respect to the quality metrics and column *F VS I*, suggest that in all but the uClinux model, the differences are significant. The differences have high effect size ($\hat{A}_{12} \geq 0.95$) on all the four models it wins. Also, the statistical results do not change by applying the Bonferroni correction. Similarly to SATIBEA, in the cases of diversity, the Filtered provides better results with respect to PFS, but worse with respect to S. However, as already mentioned, this does not indicate that IBEA is better.

Conclusively, both proposed approaches are better than the current state-of-the-art. Noticeable is the fact that SATIBEA wins the current state-of-the-art in all the employed quality metrics with maximal effect size ($\hat{A}_{12} = 1.0$). In addition, when a more heavily constrained is considered, SATIBEA performs much better than the state-of-the-art.

C. Comparing SATIBEA with Filtered, answering RQ3

Our results indicate that SATIBEA wins the Filtered method in all the model according to the HV metric. For instance, for FreeBSD, the median HV achieved by SATIBEA is almost twice the one of Filtered, i.e., ≈0.25 VS ≈0.13. These results are also statistically significant, both with and without Bonferroni correction, with a relatively high effect size (above $\hat{A}_{12} = 0.95$) on all the case. According to the Epsilon and IGD metrics SATIBEA wins in all the models, with statistical significance, except from the Fiasco where they are approximately equal. Regarding the effect sizes of the diversity measures, the two approaches are comparable with SATIBEA having a slight advantage. With respect to S, SATIBEA has a big difference in two cases, a medium difference in one and it looses or it is equal in two cases. With respect to PFS, it wins in three cases and looses in two. Therefore, since diversity is not so important if we do not have quality, overall, our results demonstrate that SATIBEA is the clear winner. It provides the best results, statistically significant, and can handle effectively heavily constrained FMs.

Conclusively, answering RQ3, our results suggest that SATIBEA is able to outperform the Filtered approach, with HV values ranging from 0.16 to 0.30.

D. Execution time of SATIBEA, answering RQ4

Our results indicate that the HV of the solutions achieved by SATIBEA converges markedly the first 15 minutes (Figure 3b). After this point, the HV increases very slowly, suggesting that SATIBEA stabilised on its ultimate solution in 15 minutes. This is an important finding since Linux is the largest available SPL hitherto reported upon in the literature. Both the smart replacement and the smart mutation operators which use the solver take less than six seconds. Thus, they help the fast convergence of the search process.

VIII. Discussion

This section discusses practical implications and threats to the validity of the findings reported in the present work.

A. Practical Implications

In the SPL context, the major challenge is the production of valid configurations. It is clear that until all constraints are satisfied, the configuration is invalid. In other words, an invalid product configuration is totally useless from practical perspective. Therefore, the effort put by the search approach in optimizing the other objectives is wasted when the resulting configuration is invalid. To investigate this, we analyze the results of the Linux FM. Specifically, we group the Pareto front solutions of IBEA according to the number of violated constraints. We visualize this situation by computing the hypervolume values when the algorithm minimizes the violated constraints. Figure 4 depicts the hypervolume achieved by the Pareto front solutions according to each number of violated constraints. We can observe that as the number of violated constraints decreases, the hypervolume also decreases. These results show that the constraints hamper the search process. Indeed, the graph clearly suggests a decreasing trend. This

is formally confirmed by a linear regression which bestows a relationship of the form $f(x) = (5.67 \times 10^{-5})x - 0.003$. This fit is good given its coefficient of determination R^2 of 0.95. In other words, our linear f explains 95% of the recorded values. Finally, it is clear that when no constraint is violated, the hypervolume is almost 0. Since the hypervolume represents how well the objectives are optimized, it shows that IBEA fails to fulfill the 4 other objectives when dealing with valid configurations. As a result, the optimization of the other objectives is limited by the minimization of the violated constraints. This explains why IBEA performs poorly compared to the Filtered approach. These results introduce the need for handling constraints independently of the search. Thus, hybrid methods like SATIBEA are the key to success.

Our evaluation focuses on large SPLs because they are typically used in industry [12], [13], [39] and they motivate the need for automation. Generally, our results show that when the number of constraints increases, the difficulty faced by the search approaches also increases. Fortunately, our results reveal that a higher number of constraints implies a higher gap between the effectiveness of the proposed and the current state-of-the-art approaches. Additionally, it should be noted that SATIBEA has an additional benefit over the state-of-the-art: it does not require any seeds. Thus, it avoids the necessary off-line computation of the seeds, which according to Sayyad *et al.* [11] consumed approximately 3 hours.

Apart from optimizing the objectives, the proposed approach can have additional applications. An interesting one is to help engineers into correct and maintain FMs. To achieve this, engineers can select multiple FM variants that represent the potential problems or changes in the original FM. Then, valid configurations (with respect to the FM variants) can be selected and evaluated towards the original FM. Such an approach can be automated as proposed by Henard *et al.* [40]. The important step here is the selection of valid configurations from the erroneous FM variants that will reflect both the potential problems and the targeted changes of the original FM. One could argue that this can be achieved with invalid configurations of the original FM. However, it is unclear how to produce invalid configurations that are both relevant and helpful in correcting the model.

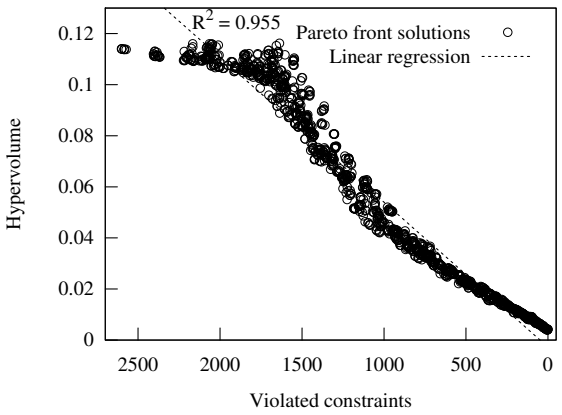


Fig. 4: Impact of the violated constraints on the hypervolume for Linux. As solutions tend to conform to the constraints of the model, the optimization of the other objectives degrades.

Finally, we note the importance of the various quality metrics. The hypervolume metric represents the extent in the optimization of user objectives. Any improvements of this metric yields a strictly better quality value [41]. In other words, a very small change in the hypervolume implies a relatively big impact in practice. Regarding the spread diversity measure, spread configurations represents solutions that are diverse in the space of the objectives, i.e., which achieve different trade-offs. Concretely, the spread configurations suggest that there are configurations in favor of each considered objective. Solutions with low spread fail to propose multiple alternative trade-offs. However, as already stated in Section VI-D diverse solutions with low quality are not meaningful in practice.

B. Threats to Validity

Several threats to the validity of the present study are identified. Our results are based on five SPLs. Hence, it is possible that our conclusions do not generalize to other cases. To reduce this threat, we took four different and large FMs with different number of features and constraints. In particular, the FMs we chose have a varied density of constraints, i.e., the number of constraints per feature vary. We used both slightly and heavily constrained models such as Linux and uClinux with an average of respectively 50 and 1.3 constraints per feature. Another threat is due to the randomness involved in the approaches studied. Indeed, there is a chance that the observed results happened by chance. To reduce this threat, we performed each approach 30 times independently, thereby reducing the influence of random effects. Another threat is identified due to potential errors, unknown parameters or differences in the implementation. In addition, the machines used may influence the results. To reduce this threat, we performed a careful verification of our results and several manual tests at all stages of our implementation. Additionally, we make publicly available both our implementation and our data. Finally, a threat is due to the artificial way the values of the attributes were assigned, i.e., the actual usage scenario, and to the replication of the state-of-the-art. Unfortunately, there is no available implementation of the previous work [12]. To overcome this issue, we carefully replicated and verified, multiple times, all parameters and the technical details of the experiments as described in [12]. Additionally, we used the same framework, algorithms and settings as the previous work.

IX. Related Work

One of the first approaches that aimed to optimize multiple configuration goals is attributed to Olaechea *et al.* [42]. This method uses a special form of FMs, called attributed FMs which record quality attributes for the features. This technique uses exact solving and consider FMs with one to three objectives. However, it fails to scale due to the computation of the exhaustive search it performs. The authors used FMs

with up to 12 features. Sayyad *et al.* [11] proposed the use of advanced multi-objective evolutionary algorithms for SPLs with five objectives to optimize. They experiment with five algorithms and conclude that IBEA is the most suitable one for the SPL context. Sayyad *et al.* were the first, to the authors knowledge, to use constraint violation as an objective for the search process. This approach was later extended by Sayyad *et al.* [12] with the aim of scaling to large SPLs. It is this technique that is investigated by the present paper, i.e., IBEA. However, as shown by our results, the proposed techniques, SATIBEA and Filtered are by far more effective.

Another approach that aims to optimize many objectives is due to Henard *et al.* [43]. In this work, a genetic algorithm is employed to generate a set of valid configurations. The approach employs a SAT solver to select valid configurations while maximizing the t-wise coverage of the selected set of configurations and taking into account other parameters such as the cost of the configurations. Despite the promising results, it fails to scale to large FMs mainly due to the expensive t-wise coverage computation [14]. In contrast, SATIBEA seeks to provide scalable solutions for the MOO problem of configuring products for SPLs. Regarding constraints handling with evolutionary approaches, several methodologies are proposed in the literature, such as including constraint knowledge in the fitness function [44], [45]. In this work, we use an external SAT solver to repair valid configurations (smart mutation operator) and to generate valid configurations (smart replacement).

Most of the existing approaches generate configurations that conform to the FM constraints while optimizing a single other objective. Benavides *et al.* [46] imposed constraints modeling extra-functional properties of the SPL features. They then applied constraint satisfaction solvers to generate all the possible configurations, the optimal ones etc. Another attempt to optimize the extra-functional properties of configurations was by White *et al.* [39]. They proposed to transform the product configuration problem into a multi-dimensional multi-choice knapsack problem to use known techniques to tackle it. White *et al.* [47] developed a tool for the multi-step configuration of evolving FMs. They show that it is possible to derive configurations automatically by mapping the SPL configuration problem to a constraint satisfaction one. Aiming at t-wise coverage, Perrouin *et al.* [48] developed a tool based on the Alloy SAT solver. Other work, e.g., [40], [49] used a SAT solver to generate valid set of configurations. Unlike the methods presented in the present paper, these methods optimize only a single objective, i.e., either the t-wise coverage or some form of attribute coverage. Furthermore, these approaches fail to scale to large FMs.

There are few techniques that scale to very large SPLs. Johansen *et al.* [50] use the covering array method based on a SAT solver to generate configuration sets that cover all the t-wise interactions between the features of an SPL. Similarly, a search-based approach that achieves scalable but partial t-wise coverage has been introduced in [14]. However, none of them targets multiple objectives. Perhaps the closest work to the present one is the one of Guo *et al.* [7]. Guo *et al.* propose the use of a genetic algorithm to tackle multiple objectives. To achieve this, it aggregates all objectives into one. This practice fails to produce a wide range of configurations and results in a single configuration that is only optimized for a specific objective weighting scheme. Additionally, it uses a repair process to make the candidate solutions valid with respect to the constraints of the FM. This process restricts the search process [11]. Since it was evaluated on artificial models and thus, it is currently unclear whether it can provide satisfactory solutions for real word FMs. Our study involves the satisfaction of multiple objectives for large, heavily constrained and real-world FMs such as the Linux kernel.

Approaches for the automated analysis of FMs have proliferated these last 20 years [19]. Such techniques enable to extract information from the FM, such as identifying the mandatory features or count the valid configurations of an SPL. These techniques rely on binary decision diagrams or solvers, such as SAT or Satisfiability Modulo Theory (SMT) solvers. The efficiency of these techniques has been investigated by Pohl *et al.* [51] with the conclusion that these approaches induce a certain overhead and that there is still room for improvement. The use of SAT solvers for reasoning on FMs has been reported as being an easy task [52]. In this work the authors conclude that the previous reports on the efficiency of SAT solvers is not incidental in practice. The FMs used in this work are extracted from existing code such as the Linux kernel. To the best of our knowledge, the efficiency of automated analysis techniques have not been investigated on such models. Finally, augmenting the features of FMs with quality attributes such as cost has been used in several previous studies, e.g., [11], [12], [42], [43], [53].

X. Conclusion and Future Work

We have demonstrated that our SPL optimization approach, SATIBEA, significantly outperforms the current state-of-the-art with maximal effect size. We also provide results that show that it is important to include constraint solving techniques in SPL optimization approaches and that our technique scales to the largest SPLs hitherto considered in the literature.

Since reproducibility has been identified as a central tenet of the research in software engineering [37], we make the source code of our implementation and our experimental data publicly available at `http://research.henard.net/SPL/ICSE_2015/`.

Future work will investigate alternative ways to improve the search process. Specifically, practices like parameter tuning [54], supervised search [55] or hybrid approaches involving both constraint-driven and genetic search will be considered.

XI. Acknowledgements

Mike Papadakis is supported by the National Research Fund, Luxembourg, INTER/MOBILITY/14/7562175. Mark Harman is supported by Engineering and Physical Sciences Research Council (the EPSRC) grants Genetic Improvement of Software for Multiple Objectives (GISMO: EP/I033688) and Dynamic Adaptive Automated Software Engineering (DAASE: EP/J017515).

REFERENCES

[1] A. Metzger and K. Pohl, "Software product line engineering and variability management: Achievements and challenges," in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. New York, NY, USA: ACM, 2014, pp. 70–84. [Online]. Available: http://doi.acm.org/10.1145/2593882.2593888

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[3] P. Knauber, J. B. Muñoz, G. Böckle, J. C. S. d. P. Leite, F. v. d. Linden, L. M. Northrop, M. Stark, and D. M. Weiss, "Quantifying product line benefits," in *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, ser. PFE '01. London, UK, UK: Springer-Verlag, 2002, pp. 155–163. [Online]. Available: http://dl.acm.org/citation.cfm?id=648114.748915

[4] M. Harman, Y. Jia, J. Krinke, B. Langdon, J. Petke, and Y. Zhang, "Search based software engineering for software product line engineering: a survey and directions for future work (keynote paper)," in 18^{th} *International Software Product Line Conference*, ser. SPLC '14, Florence, Italy, September 2014, to appear.

[5] J. White, B. Doughtery, and D. C. Schmidt, "Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints," in *12th International Conference on Software Product Lines*, Sep. 2008, pp. 209–216.

[6] J. Li, X. Liu, Y. Wang, and J. Guo, "Formalizing feature selection problem in software product lines using 0-1 programming," in *6th International Conference on Intelligent Systems and Knowledge Engineering*, 2011.

[7] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," *The Journal of Systems and Software*, vol. 84, no. 12, pp. 2208–2221, Dec. 2011.

[8] Z. Wu, J. Tang, C. K. Kwong, and C. Y. Chan, "An optimization model for reuse scenario selection considering reliability and cost in software product line development," *International Journal of Information Technology & Decision Making*, vol. 10, no. 5, pp. 811–841, 2011.

[9] A. S. Sayyad, K. Goseva-Popstojanova, T. Menzies, and H. Ammar, "On parameter tuning in search-based software engineering: A replicated empirical study," in *International Workshop on Replication in Empirical Software Engineering Research*, Oct. 2013.

[10] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Optimum feature selection in software product lines: Let your model and values guide your search," in *1st International Workshop on Combining Modelling and Search-Based Software Engineering*, 2013, pp. 22–27.

[11] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 492–501. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486853

[12] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *28th International Conference on Automated Software Engineering*, 2013, pp. 465–474.

[13] F. Loesch and E. Ploedereder, "Optimization of variability in software product lines," in *11th International Software Product Line Conference*, 2007, pp. 151–162.

[14] 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 configurations for software product lines," *IEEE Trans. Software Eng.*, vol. 40, no. 7, pp. 650–670, 2014.

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

[16] J. Coplien, D. Hoffman, and D. M. Weiss, "Commonality and variability in software engineering," *IEEE Software*, vol. 15, no. 6, pp. 37–45, 1998.

[17] K. C. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," *IEEE Software*, vol. 19, no. 4, pp. 58–65, 2002.

[18] T. Thum, D. Batory, and C. Kastner, "Reasoning about edits to feature models," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 254–264. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070526

[19] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.is.2010.01.001

[20] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, ser. VaMoS '13. New York, NY, USA: ACM, 2013, pp. 7:1–7:8. [Online]. Available: http://doi.acm.org/10.1145/2430502.2430513

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

[22] E. Zitzler and S. Knzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science, X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervs, J. Bullinaria, J. Rowe, P. Tio, A. Kabn, and H.-P. Schwefel, Eds., vol. 3242. Springer Berlin Heidelberg, 2004, pp. 832–842. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30217-9_84

[23] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki, "Comparison of exact and approximate multi-objective optimization for software product lines," in 18^{th} *International Software Product Line Conference*.

[24] R. Olaechea, "Optimization of variability in software product lines," Master's thesis, University of Waterloo, Ontario, 2013.

[25] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 461–470. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985856

[26] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "Variability modeling in the systems software domain," Generative Software Development Laboratory, University of Waterloo, Technical Report, 2012.

[27] ——, "Variability modeling in the real: A perspective from the operating systems domain," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 73–82. [Online]. Available: http://doi.acm.org/10.1145/1858996.1859010

[28] D. L. Berre and A. Parrain, "The sat4j library, release 2.2," *JSAT*, vol. 7, no. 2-3, pp. 59–6, 2010.

[29] J. J. Durillo and A. J. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[30] J. Knowles, L. Thiele, and E. Zitzler, "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers," Computer Engineering and Networks Laboratory (TIK), ETH Zurich, TIK Report 214, Feb. 2006.

[31] D. Brockhoff, T. Friedrich, and F. Neumann, "Analyzing hypervolume indicator based algorithms," in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature: PPSN X*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 651–660. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87700-4_65

[32] T. Hawkins, *Lebesgue's theory of integration: its origins and development*. American Mathematical Soc., 2001, vol. 282.

[33] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Trans. Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2003.810758

[34] G. B. L. David A. Van Veldhuizen, "Multiobjective evolutionary algorithm research: A history and analysis," Tech. Rep.

[35] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[36] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985795

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

[38] A. Vargha and H. D. Delaney, "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong," *Journal on Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[39] J. White, B. Dougherty, and D. C. Schmidt, "Selecting highly optimal architectural feature sets with filtered cartesian flattening," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1268–1284, 2009.

[40] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Assessing software product line testing via model-based mutation: An application to similarity testing," in *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, ser. ICSTW '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 188–197. [Online]. Available: http://dx.doi.org/10.1109/ICSTW.2013.30

[41] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration," in *Proceedings of the 4th International Conference on Evolutionary Multi-criterion Optimization*, ser. EMO'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 862–876. [Online]. Available: http://dl.acm.org/citation.cfm?id=1762545.1762618

[42] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside, "Modelling and multi-objective optimization of quality attributes in variability-rich software," in *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, ser. NFPinDSML '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2420942.2420944

[43] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Multi-objective test generation for software product lines," in *Proceedings of the 17th International Software Product Line Conference*, ser. SPLC '13. New York, NY, USA: ACM, 2013, pp. 62–71. [Online]. Available: http://doi.acm.org/10.1145/2491627.2491635

[44] B. G. W. Craenen, A. E. Eiben, and E. Marchiori, "How to handle constraints with evolutionary algorithms," in *Practical Handbook of Genetic Algorithms*. Chapman & Hall/CRC, 2001, pp. 341–361.

[45] D. Reid, "Genetic algorithms in constrained optimization," *Mathematical and Computer Modelling*, vol. 23.

[46] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, ser. CAiSE'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 491–503. [Online]. Available: http://dx.doi.org/10.1007/11431855_34

[47] J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, and D. C. Schmidt, "Evolving feature model configurations in software product lines," *Journal of Systems and Software*, vol. 87, pp. 119–136, 2014.

[48] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. l. Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ser. ICST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 459–468. [Online]. Available: http://dx.doi.org/10.1109/ICST.2010.43

[49] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon, "Towards automated testing and fixing of re-engineered feature models," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1245–1248. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486975

[50] M. F. Johansen, O. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *Proceedings of the 16th International Software Product Line Conference - Volume 1*, ser. SPLC '12. New York, NY, USA: ACM, 2012, pp. 46–55. [Online]. Available: http://doi.acm.org/10.1145/2362536.2362547

[51] R. Pohl, K. Lauenroth, and K. Pohl, "A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 313–322. [Online]. Available: http://dx.doi.org/10.1109/ASE.2011.6100068

[52] M. Mendonca, A. Wasowski, and K. Czarnecki, "Sat-based analysis of feature models is easy," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 231–240. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753235.1753267

[53] G. Zhang, H. Ye, and Y. Lin, "Using knowledge-based systems to manage quality attributes in software product lines," in *Proceedings of the 15th International Software Product Line Conference, Volume 2*, ser. SPLC '11. New York, NY, USA: ACM, 2011, pp. 32:1–32:7. [Online]. Available: http://doi.acm.org/10.1145/2019136.2019172

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

[55] Y. Jia, M. B. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction testing strategies using hyperheuristic search," *RN/UCL*, vol. 13, p. 17, 2013.