Scalable Product Line Configuration: A Straw to Break the Camel's Back

Abdel Salam Sayyad Joseph Ingram Tim Menzies Hany Ammar
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV, USA
{asayyad, jingram3}@mix.wvu.edu tim@menzies.us hany.ammar@mail.wvu.edu

***Abstract*—Software product lines are hard to configure. Techniques that work for medium sized product lines fail for much larger product lines such as the Linux kernel with 6000+ features. This paper presents simple heuristics that help the Indicator-Based Evolutionary Algorithm (IBEA) in finding sound and optimum configurations of very large variability models in the presence of competing objectives. We employ a combination of static and evolutionary learning of model structure, in addition to utilizing a pre-computed solution used as a "seed" in the midst of a randomly-generated initial population. The seed solution works like a single straw that is enough to break the camel's back –given that it is a feature-rich seed. We show promising results where we can find 30 sound solutions for configuring upward of 6000 features within 30 minutes.**

***Index Terms*—Variability models, automated configuration, multiobjective optimization, evolutionary algorithms, SMT solvers.**

I. Introduction

A. Motivation

Scalability of Search-Based Software Engineering (SBSE) methods is of high importance because it can mean the difference between theoretical obscurity and industrial adoption. The larger and more complex the application examples are, the closer they resemble practical applications, and the more believable the result will be. Yet the lack of scalability of results is one of the biggest problems facing software engineers, according to Harman et al. [12]. They state that: "Many approaches that are attractive and elegant in the laboratory turn out to be inapplicable in the field, because they lack scalability."

A case in point is the subject of this paper: the many-objective optimum feature selection in software product lines. Many results in the automated analysis of software product lines were validated using feature models published in online feature model repositories such as SPLOT [17]. Examples are: Pohl et al. [20], Lopez-Herrejon and Egyed [16], Johansen et al. [14], Mendonca et al. [19] and our own previous work [23] [22]. Most of the feature models in SPLOT were produced for academic purposes without representing actual systems. One such model is "Electronic Shopping," designed by Lau [15], the largest in SPLOT with 290 features. While it might be a "best effort" in emulating a real system, it does not represent an actual project. Berger et al. [5] explain in detail the differences in properties between SPLOT feature models and the large feature models that they developed by reverse-engineering real systems, and published in the LVAT (Linux Variability Analysis Tools) repository[1]. In short, SPLOT models had significantly smaller and less constrained models with lower branching factors, but they also had higher ratios of feature groups and deeper leaves than LVAT models. This shows an underlying gap between academic assumptions and actual properties of software product lines.

The only two studies we know that experimented with the LVAT feature models were done by Johansen et al. [14] who generated test covering arrays for feature models, and Henard et al. [13] who worked on prioritizing t-wise test suites. Both experimented with three very large models from the LVAT repository (Linux, eCos, and FreeBSD), in addition to models from SPLOT and other sources. Our work is the first to attempt the many objective optimization of product line configuration.

Other researchers attempted to prove scalability of their methods using randomly-generated feature models that followed a set of assumed characteristics. Examples are: White et al. [30] [29] [28] [31], Shi et al. [25], Guo et al. [10], and Mendonca et al. [19]. While those randomly-generated models can be larger in size than published models, they still suffer from the same assumptions that diverge from the properties of real systems.

In this work, we seek to show the scalability of 5-objective optimization of software product lines using the Indicator-Based Evolutionary Algorithm (IBEA) compared with the Nondominated Sorting Genetic Algorithm-II (NSGA-II). Previously [23], we applied IBEA and NSGA-II –among other algorithms– to feature models from SPLOT, with sizes ranging from 43 to 290 features. With IBEA, we were able to achieve 5-objective optimization with a significant amount of fully-correct configurations. Other algorithms, including NSGA-II, failed to learn the model constraints, and produced few usable solutions. In this paper, we apply both IBEA and NSGA-II to 7 large models from LVAT. In 6 out of 7 models, we successfully show the superiority of IBEA over NSGA-II in producing significant amounts of fully correct and highly optimal configurations. Towards that end, IBEA is assisted by

[1] https://code.google.com/p/linux-variability-analysis-tools/source/browse/?repo=formulas

a static scan to detect features that must have fixed values, and those features are excluded from the evolutionary process.

For the largest model in this experiment, i.e. Linux kernel, the above combination of static and evolutionary learning did not succeed in producing any correct configurations during the first 30 minutes. To address this problem, we resorted to a novel approach, in which we pre-computed one correct configuration and planted it in the initial population (of 300 candidate solutions) for the 5-objective optimization. The result was 30 correct configurations in the first 30 minutes of the 5-objective optimization. When we planted 30 correct seeds, the 30-minute result was the same, i.e. about 30 valid solutions. Therefore the effect of one carefully selected seed was enough to influence a randomly-generated population into finding a range of solutions that can be suggested to the user in the initial stage of interactive configuration. The seeding approach is depicted in Fig. 1.

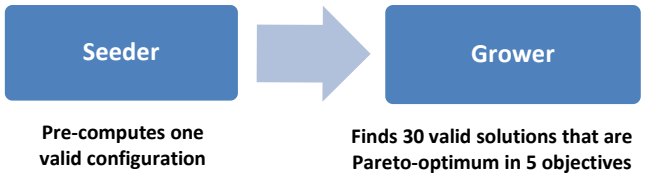

Fig. 1. Depiction of the seeding approach

One caveat is that the magic "seed" has to be feature-rich, i.e. it must be a solution in which a sufficiently large number of features are enabled. The result we obtain points us to the following guideline:

"The proper selection of (a) seed(s) in the initial population is a key to scalability of product line configuration."

B. Contributions of this Paper

This paper makes the following contributions:

1- A proof of scale-up of previous results [23] [22] that showed superiority of IBEA over other MEOAs (e.g. NSGA-II) that used absolute dominance criteria coupled with diversification measures in the fitness assignment. IBEA uses a continuous dominance metric that makes better use of the user preferences.

2- First-time attempt to automate the configuration of the very large variability models available from LVAT feature model repository which resulted from the works of Berger et al. [5] [24] [4] [3]. We are not aware of any previous work that used those models for automated configuration. Johansen et al. [14] experimented with generating test covering arrays for three models (Linux, eCos, and FreeBSD), but their tools were not able to handle the large sizes of these models for most of the purposes of their experiment. Henard et al. [13] presented a search-based technique for prioritizing t-wise test suites, and applied it to eCos, FreeBSD, and Linux with good results.

3- A novel approach that relies on IBEA's ability to exploit user preference knowledge in reaching optimum results, but also enlists the help of a pre-computed correct solution (a seed) and a static analysis of the model structure to aid IBEA in converging faster to a large number of correct configurations. The seeding approach was used by Fraser and Arcuri [9] in the context of search-based software testing.

4- A breakthrough scalability result. Using the novel approach in 3 (above), we now show that it is possible to configure product lines as large as 6000+ features. Note that this is a significant improvement in the state of the art in this area since prior work was shown to configure up to 290 features only.

C. Organization of this Paper

Section II discusses related work in the automated analysis of feature models. Section III introduces background material on feature modeling, MEOAs and the Z3 SMT solver. Section IV explains the experimental setup, and section V presents the results. We discuss the findings and their impact in section VI. In section VII we discuss potential threats to validity, and then in section VIII we offer our conclusions and directions for future work.

II. Related Work

First, we discuss related work in the area of automated product configuration and feature selection.

The idea of extending (or augmenting) feature models with quality attributes was proposed by many, among them Zhang et al. [32]. The following papers used a similar approach and synthetic data to experiment with optimizing feature selection in SPLs.

Soltani et al. [26] employed Hierarchical Task Network (HTN) planning, a popular planning technique to automatically select suitable features that satisfy the stakeholders' business concerns and resource limitations. A performance evaluation was provided with three feature models containing 25, 45, and 65 features. The worst case run time was reported to be 89 seconds, which is significant for these small-size feature models.

Benavides et al. [1] provided automated reasoning on extended feature models. They assigned extra-functionality such as price range or time range to features. They modeled the problem as a Constraint Satisfaction Problem, and solved it using CSP solvers to return a set of features which satisfy the stakeholders' criteria.

White et al. [28] mapped the feature selection problem to a multidimensional multi-choice knapsack problem (MMKP). They apply Filtered Cartesian Flattening to provide partially optimal feature configuration.

Also White et al. [29] introduced the MUSCLE tool which provided a formal model for multistep configuration and mapped it to constraint satisfaction problems (CSPs). Hence, CSP solvers were used to determine the path from the start of the configuration to the desired final configuration. Non-functional requirements were considered such as cost constraints between two configurations. A sequence of minimal feature adaptations is calculated to reach from the initial to the desired feature model configurations.

The limitations of these methods are obvious, given the small models that they experimented with. As SPLs become larger the problem grows more intractable. More recently, a Genetic Algorithm was used to tackle this problem [10]. Although the problem is obviously multiobjective, the various objectives where aggregated into one and a simple GA was used. The result is to provide the product manager with only one "optimal" configuration, which is only optimal according to the weights chosen in the objective formula. Also, they used a repair operator to keep all candidate solutions in line with the feature model all throughout the evolutionary process.

Next, we discuss other related work in the general area of automated analysis of feature models.

In [20], a large experiment was performed to measure the efficiency of available BDD, SAT and CSP solvers to perform four analysis operations on 90 feature models from the SPLOT repository. They reported long run times for certain operations, and they cancelled certain runs with the larger feature models when the run time exceeded three hours. An exponential runtime increase with the number of features for non-BDD solvers on the "valid" operation was also reported.

In [16], a basic search method (Breadth-First Search) is used to find feature model inconsistencies and suggest fixing sets. The method was run with 60 feature models from the SPLOT website, the largest being 94 features. They report that computation time increases steadily as the number of features increases (with 94 features it took 1600 sec. approx.).

In [19], efficient ordering heuristics are proposed for BDDs that represent feature models. Such ordering can dramatically reduce the size of BDDs, thus allowing fast processing for interactive configuration algorithms. The proposed heuristics were tested with five realistic feature models, in addition to randomly-generated feature models with larger sizes. It was shown that the heuristics produce high quality variable orders that enable the compilation of large feature models with up to 2,000 features.

In [18], it is shown that the task of satisfiability (SAT) solving of realistic models is easy. In particular, the phenomenon of phase transition is not observed for realistic feature models. The explanation for this is that many real world problems are either over-constrained (in terms of variability: they have no realizable products) or under-constrained (they have many easily identifiable realizations). For instance, consistency checks on randomly-generated models with up to 10,000 features and a large number of cross-tree constraints took about 0.4 seconds. In addition, computing valid domains was completed in about 22 seconds for models with 5,000 features and a fairly large number of cross-tree constraints.

In all the works mentioned above, the testing was done with relatively small feature models published in academic repositories like SPLOT, or with large feature model that were randomly-generated based on the same characteristics as SPLOT models. Large feature models that represent actual code (such as those published in LVAT) have not yet been used to test out automated analysis and configuration methods.

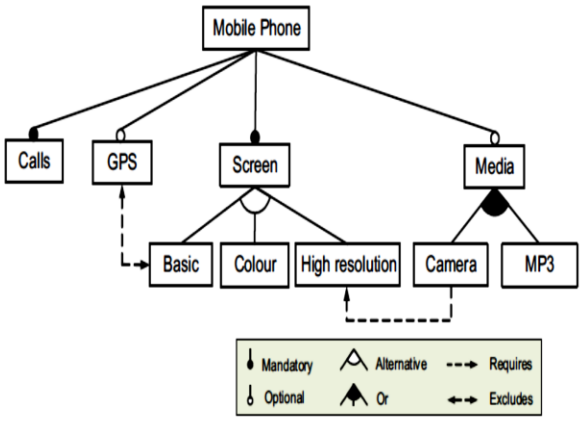

Fig. 2. Feature model for mobile phone product line *[2]*

FM = **(Mobile Phone ↔ Calls)**
∧ (Mobile Phone ↔ Screen)
∧ (GPS → Mobile Phone)
∧ (Media → Mobile Phone)
∧ (Screen ↔ XOR (Basic, Color, High resolution))
∧ (Media ↔ Camera ∨ MP3)
∧ (Camera → High resolution)
∧ ¬(GPS ∧ Basic)

Fig. 3. Mobile phone feature model as a Boolean expression

III. BACKGROUND

A. Feature Models

A feature is an end-user-visible behavior of a software product that is of interest to some stakeholder. A feature model represents the information of all possible products of a software product line in terms of features and relationships among them. Feature models are a special type of information model widely used in software product line engineering. A feature model is represented as a hierarchically arranged set of features composed by:

1- Relationships between a parent feature and its child features (or subfeatures).
2- Cross-tree constraints that are typically inclusion or exclusion statements in the form: if feature F is included, then features A and B must also be included (or excluded).

Figure 2, adapted from [2], depicts a simplified feature model inspired by the mobile phone industry.

The full set of rules in a feature model can be captured in a Boolean expression, such as the one in Fig. 3, which shows the expression for the mobile phone feature model. From it we can conclude that the total number of rules in this feature model is 16, including the following:

- The root feature is mandatory.
- Every child requires its own parent.
- If the child is mandatory, the parent requires the child.
- Every group adds a rule about how many members can be chosen.
- Every cross-tree constraint (CTC) is a rule.

The feature models used in this study were obtained from the Linux Variability Analysis Tools (LVAT) feature model repository, which resulted from the works of Berger et al. [5] [24] [4] [3]. The models were reverse-engineered from open-source code, comments, and documentation of such projects as the Linux kernel, eCos and FreeBSD operating systems, and other large projects. The resulting feature models had distinctly different properties than models published by academic researchers, such as those in SPLOT [17]. The LVAT models are significantly larger in size, more constrained, and have higher branching factors than academic models, but they also had lower ratios of feature groups and, in general, shallower leaves. The LVAT models provide an opportunity for testing the scalability of many results in feature modeling for software product lines.

The models downloaded from LVAT website had the DIMACS format, which expresses each model as a formula in the Conjunctive Normal Form (CNF).

B. Multiobjective Optimization

Many real-world problems involve simultaneous optimization of several incommensurable and often competing objectives. Often, there is no single optimal solution, but rather a set of alternative solutions. These solutions are optimal in the wider sense that no other solutions in the search space are superior to them when all objectives are considered [34].

Formally, a *vector* $u = \{u_1, \ldots, u_k\}$ is said to be dominated by a vector $v = \{v_1, \ldots, v_k\}$ if and only if u is partially less than v, i.e.

$$\forall i \in \{1, \ldots, k\}, u_i \leq v_i \; and \; \exists \, i \in \{1, \ldots, k\} : u_i < v_i \quad (1)$$

The set of all points in the objective space that are not dominated by any other points is called the *Pareto Front*.

C. Multiobjective Evolutionary Optimization Algorithms (MEOAs)

Many algorithms have been suggested over the past two decades for multiobjective optimization based on evolutionary algorithms that were designed primarily for single-objective optimization, most notably Genetic Algorithms.

We have previously experimented [23] with MEOAs that are implemented in the jMetal framework [8] as applied to SPLOT feature models. We found a remarkable advantage in performance for the Indicator-Based Evolutionary Algorithm (IBEA) [33], as compared to Pareto-based algorithms, of which the Nondominated Sorting Genetic Algorithm, version 2 (NSGA-II) [7] is the best known.

The fundamental difference between these two types of algorithms is in the ranking criterion (i.e. fitness assignment) used to determine which individuals have stronger chance to survive to the next generation. Thus we focus on the ranking criteria in both algorithms.

1) Nondominated Sorting Genetic Algorithm, version 2 (NSGA-II):

The sorting procedure in NSGA-II is depicted in Fig. 4, taken from [7]. It shows how the combined primary and secondary population gets sorted according to domination, where F_1 contains all nondominated solutions; F_2 contains all nondominated solutions after excluding F_1 and so on. When the solutions within F_3 need to be sorted for truncation, they are ranked according to crowding distance, a value calculated from distances to nearest neighbors in all objective values. Thus diversity preservation is the second criterion –after domination– to determine fitness for survival.

2) Indicator-Based Evolutionary Algorithm (IBEA):

Figure 5 provides an outline of the IBEA algorithm. The details can be found in [33].

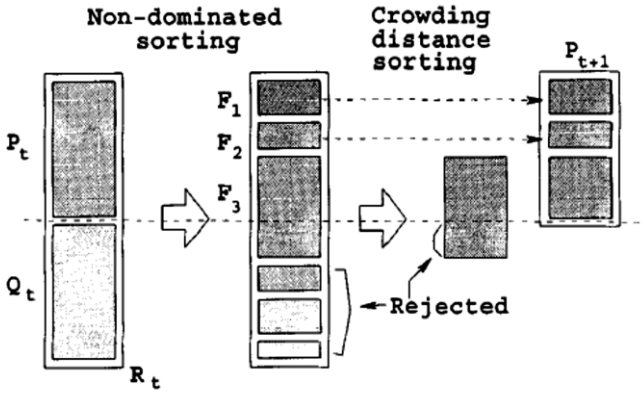

Fig. 4. NSGA-II sorting procedure *[7]*

Input: α *(population size)*
N *(maximum number of generations)*
κ *(fitness scaling factor)*
Output: A *(Pareto set approximation)*

Step 1: ***Initialization****: Generate an initial population P of size α; and an initial mating pool P' of size α; append P' to P; set the generation counter m to 0.*
Step 2: ***Fitness assignment****: Calculate fitness values of individuals in P, i.e., for all* $\boldsymbol{x}1 \in P$ *set*

$$F(x_1) = \sum_{x_2 \in P \setminus \{x_1\}} -e^{-I(\{x_2\},\{x_1\})/\kappa} \quad (2)$$

Where I(.) is a dominance-preserving binary indicator.
Step 3: ***Environmental selection****: Iterate the following three steps until the size of population P does not exceed α:*

1. Choose an individual $\boldsymbol{x}* \in P$ *with the smallest fitness value, i.e.,* $F(\boldsymbol{x}*) \leq F(\boldsymbol{x})$ *for all* $\boldsymbol{x} \in P$.

2. Remove $\boldsymbol{x}*$ *from the population.*

3. Update the fitness values of the remaining individuals, i.e. $F(\boldsymbol{x}) = F(\boldsymbol{x}) + e^{-I(\{x_2\},\{x_1\})/\kappa}$ *for all* $\boldsymbol{x} \in P$.

Step 4: ***Termination****: If $m \geq N$ or another stopping criterion is satisfied then set A to the set of decision vectors represented by the nondominated individuals in P. Stop.*
Step 5: ***Mating selection****: Perform binary tournament selection with replacement on P in order to fill the temporary mating pool P'.*
Step 6: ***Variation****: Apply recombination and mutation operators to the mating pool P' and add the resulting offspring to P. Increment the generation counter ($m = m + 1$) and go to Step 2.*

Fig. 5. Outline of IBEA *[33]*

Equation 2 in Fig. 5 shows IBEA's fitness assignment. Each solution is given a weight based on I(.), a dominance-preserving quality indicator, thus factoring in more of the optimization objectives of the user. The authors of IBEA, Zitzler and Kunzli, designed the algorithm such that "preference information of the decision maker" can be "integrated into multiobjective search" [33]. It is noticed here that the ranking criteria in IBEA place no emphasis on diversity of solutions, thus diverging from the conventional trend set by NSGA-II, and followed by many others.

This difference in ranking criteria causes IBEA to outperform NSGA-II when the objective space increases in dimension. In [27], it is experimentally demonstrated with real-valued test functions that the performance of NSGA-II and SPEA2 rapidly deteriorates with increasing dimension, and that other algorithms (such as IBEA) cope very well with high-dimensional objective spaces. It is argued that NSGA-II tends to "increase the distance to the Pareto front in the first generations because the diversity-based selection criteria favor higher distances between solutions. Special emphasis is given to extremal solutions with values near zero in one or more objectives. These solutions remain non-dominated and the distance cannot be reduced thereafter."

D. Z3 SMT Solver

We will use the Z3 SMT solver as one way to generate a known correct configuration for the large feature models. This method is fast and straightforward since the models are already expressed in DIMACS format, which is a direct representation of the model's Boolean formula, such as the one in Fig. 3.

Satisfiability modulo theories (SMT) generalizes Boolean satisfiability (SAT) by adding equality reasoning, arithmetic, fixed-size bit-vectors, arrays, quantifiers, and other useful first-order theories. An SMT solver is a tool for deciding the satisfiability (i.e. validity) of formulas in these theories. Z3 is an efficient SMT Solver freely available from Microsoft Research. It is used in various software verification and analysis applications. [6]

Although the configuration generated by Z3 was not useful in this experiment compared to the one generated using IBEA, we still plan to use Z3 in future work to efficiently find a set of correct configurations that would help IBEA in converging faster to a larger set of sound and optimal solutions.

IV. SETUP

A. Feature Models Used in this Study

The LVAT formula repository includes 15 models (as of May 2013), each represented in two formats: Boolean and DIMACS. Table I lists the 7 models for which the results in this study are reported.

B. Feature Attributes

Our research explores alternate methods to explore complex decision spaces. In our recent literature review [21], we found that most MEOA research in software engineering explores a very simplistic two-valued objective space.

TABLE I. MODELS USED IN THIS STUDY

Model	Version	Features	Ref.
ToyBox	0.1.0	544	[5]
axTLS	1.2.7	684	[5]
eCos	3.0	1244	[24], [4]
FreeBSD	8.0.0	1396	[24], [3]
Fiasco	2011081207	1638	[5]
uClinux	20100825	1850	[5]
Linux X86	2.6.28	6888	[24], [3]

In our work with users, we find that merely exploring two objectives is insufficient to capture the breadth of their concerns. Therefore, when we certify different optimizers, we take care to explore problems with up to half a dozen objectives. To make such rich objective spaces, we augment simpler models with a rich set of objectives. Specifically, we augmented the feature models with 3 attributes for each feature: *COST*, *USED_BEFORE*, and *DEFECTS*. The values were selected stochastically according to distributions that emulate software projects. COST takes real values distributed normally between 5.0 and 15.0, USED_BEFORE takes Boolean values distributed uniformly, and DEFECTS takes integer values distributed normally between 0 and 10.

The only dependency among these qualities is:

if (not USED_BEFORE) then DEFECTS = 0 (3)

C. Problem Representation

The feature models were represented as binary strings, where the number of bits is equal to the number of features. If the bit value is TRUE then the feature is selected, otherwise the feature is removed (i.e. deselected).

D. Problem Formulation; Defining the Optimization Objectives

In this work we optimize the following objectives:

1- Correctness; i.e. compliance to the relationships and constraints defined in the feature model. Since jMetal treats all optimization objectives as minimization objectives, we seek to minimize rule violations.
2- Richness of features; we seek to minimize the number of deselected features.
3- Features that were used before; we seek to minimize the number of features that weren't used before.
4- Known defects; which we seek to minimize.
5- Cost; which we seek to minimize.

The second objective (i.e. richness of features) counteracts the effects of the other objectives by increasing the number of selected features while minimizing violations, defects, and cost. Without it, the final Pareto front would crowd in the area with minimum features and thus would provide a narrow set of options to the end user.

E. MEOA Parameters

The following parameter values were used after rudimentary runs for parameter tuning. It is noted that low values for crossover and mutation rates perform better with feature models, as we found in previous work [22].

TABLE II. PARAMETER VALUES

Parameter	Value
Population size	300
Crossover rate	0.05
Mutation rate	0.001
Run time	30 minutes
Independent runs	10

F. Run Time as Stopping Criterion

In [23], we compared MEOAs by allowing each to perform a fixed number of fitness function evaluations, which is a commonly used approach. The number of evaluations is proportional to the total run time and the required CPU power. Yet, the total run time is affected by many other algorithm-dependent operations, including the fitness ranking of individuals in each generation. This leads to varying runtimes with the same number of evaluations. For instance, we noticed that IBEA took five times longer than NSGA-II to perform the same number of evaluations, which meant that IBEA spent far more time in fitness ranking than NSGA-II. This is expected from our study of fitness ranking criterion in subsection III.C.

The question here is: which criterion shall we fix in order to have a fair comparison among algorithms? We have come to the opinion that each algorithm should be given a fixed amount of time to calculate its best approximation of the Pareto front. A better algorithm should score better on the quality indicators (HV, %correct) within that duration of time. Going back to the comparison between IBEA and NSGA-II, if both are given the same duration of time, then NSGA-II would perform far more evaluations than IBEA, and thus would be given a better chance to improve its results. As we will see in the coming section, providing NSGA-II with the chance to evolve more generations did not help it to overcome IBEA at producing more correct solutions or better HV.

In addition, the user should be more concerned with the amount of time it takes to optimize, than with the number of evaluations. CPU power is often available at the user's disposal, and the algorithms should utilize that CPU power to produce the best results in the least amount of time, regardless of number of evaluations or number of evolved generations.

Therefore, in the this paper's experiments we make our comparisons of the results after limiting the amount of time given to each algorithm to 30 minutes, regardless of number of evaluations each algorithm were able to perform.

G. Quality of Pareto Front

We compare the performance of MEOAs using the following quality indicators:

1- Hypervolume (HV): defined in [34], is a measure of the size of the space covered underneath the Pareto front. If the objectives are all to be maximized, then the preferred Pareto front is the one with the highest Hypervolume. In jMetal, all objectives are minimized, but the Pareto front is inverted before calculating hypervolume, thus the higher the hypervolume the closer to optimum the Pareto front is.

2- %Correct: i.e. the percentage of fully-correct solutions, which is an indicator particular to this problem. Since correctness is an optimization objective that evolves over time, there maybe points in the final Pareto front that have rule violations. Such points are not likely to be useful to the user. We are interested in percentage of points within the Pareto front that have zero violations, and thus a full-correctness score.

3- TT50%: i.e. Time to achieve 50% correct solutions is another problem-specific indicator that we added as a measure of the speed of convergence to a large amount of valid solutions. This is a useful comparison figure when the final %Correct value is the same, since it shows who arrived faster at the 50% milestone.

V. RESULTS

In the following, we run the 5-objective optimization problem that we described in III.D. We first try NSGA-II and IBEA without adding any knowledge of model constraints, and then we add the "feature fixing" technique, which we find to help IBEA in optimizing the configuration of 6 large models from LVAT within the allocated 30 minutes. As for the 7th and largest model, i.e. Linux kernel, IBEA requires further domain-knowledge assistance, which we offer in the form of a seed planted in the initial population. This technique results in finding 30 correct configurations in 30 minutes.

A. Static Analysis to Detect Fixed Features

Our original approach to the configuration of feature models [23] was to start from a population of randomly generated configurations, and let the evolutionary process promote those configurations that conform to the feature model. That approach worked well for the small feature models in SPLOT, although with extended run times, but it was clear that we needed to guide the evolutionary algorithms to closely respect the structure of feature models.

In the DIMACS formulas representing our feature models, certain disjunctions (rules) only include one feature, which means that the feature is either mandatory (a commonality) which must always be selected, or a dead feature which must always be deselected. Also, we looked for disjunctions (rules) that included two features but one of them was fixed in the first round, and thus the second one was fixed as well.

Once a feature is detected as fixed, we fix it in the initial population, while all other features are subject to random configuration, and we restrict the bit mutation operator to only flipping features that are not fixed.

Table III shows the amount of fixed features detected in each model. It also shows the amount of "skipped rules", i.e. the rules that we stop checking in our fitness evaluation since they only include fixed features. We observe that eCos, FreeBSD, and the Linux X86 models have few fixed features.

Table IV shows the results comparing IBEA and NSGA-II with and without feature fixing. Each algorithm is run 10 times for 30 minutes in each case. The median values are reported. We also performed Mann-Whitney tests to assess the statistical significance of the %Correct indicator. **We highlight the %Correct in bold if the confidence level exceeds 95% when comparing each method to the one to its left.**

TABLE III. FIXED FEATURES AND SKIPPED RULES

Model	Total Features	Fixed Features	Total Rules	Skipped Rules
ToyBox	544	363	1020	394
axTLS	684	384	2155	259
eCos	1244	19	3146	11
FreeBSD	1396	3	62183	20
Fiasco	1638	995	5228	553
uClinux	1850	1244	2468	1850
Linux X86	6888	94	343944	699

We make the following observations:

1- The feature fixing approach is still not enough for the largest model, the Linux kernel. There were no valid solutions after 30 minutes for all cases.

2- IBEA outperforms NSGA-II in terms of the %Correct indicator. Feature fixing helps NSGA-II achieve better %Correct and HV, but the majority of solutions remain useless due to violations of the model constraints. This confirms previous findings by the authors [23] regarding the superiority of a continuous measure of domination (as in IBEA) over absolute dominance used in NSGA-II (see subsection III.C).

3- IBEA with feature fixing achieves remarkable results for six models (the numbers highlighted in bold). For two of these six models (ToyBox and uClinux), the percentage of correct solutions is 25% and 31% respectively. When considering that the final Pareto front is composed of 300 individuals, 25% corresponds to 75 fully-correct solutions, and 31% means 93 valid solutions. This is remarkable as well compared to NSGA-II or IBEA without feature fixing.

4- IBEA without feature fixing achieved high %Correct with two models (eCos and FreeBSD). When feature fixing was used, the TT50% indicator showed a faster growth of correct configurations, while the HV indicator showed an improvement in the overall optimality of solutions.

5- Some cases show a lower HV value when the %Correct value is improved. This means that, when the number of violations is high, the other 4 objectives take closer to optimum values, which would not be useful because of the rule violations.

B. Using a Pre-Computed Correct Solution as Seed to IBEA

The results in the previous part are encouraging, but it's clear that IBEA needs more assistance to achieve acceptable configurations for the Linux model within reasonable time.

Our next innovative technique was to pre-compute a correct configuration and plant it like a seed in the initial population of the evolutionary algorithms. The intuition behind this was that the randomly-generated members of the initial population are highly likely to violate thousands of feature model rules and be punished for that in the fitness assignments. When an individual in the initial population stands out as a fully-correct solution, then it should be promoted more often than others for crossover with other individuals, and would survive through successive generations due to elitism. Thus the "seed" acts as a role model to the "chaotic" members of the population. This technique proved to be useful as we will see next.

First, we present two different ways of pre-computing a correct solution:

1- Using the Z3 SMT solver. Z3 takes the DIMACS formula as input, and outputs the first correct solutions that it finds. This technique is fast, but it tends to produce correct solutions with a low number of 1's; i.e. a low number of selected features.

2- Using 2-objective optimization with IBEA, where one objective is to minimize rule violations, while the other objective is to maximize the number of selected features. This technique can be time-consuming for very large feature models, but it produces more selected features.

Table V shows the time it took each of these two techniques to generate a correct solution, and the number of selected features within that solution.

Notice that the eCos model accepts the "zero-feature" solution, which definitely is a bug in that formula.

To show the benefit of the "seeding" technique, we apply IBEA, with feature fixing, and seeding to the Linux X86 feature model, with the full 5 optimization objectives. Three different kinds of seeds are tried separately:

1- One "feature-rich" seed generated using 2-objective IBEA, along with 299 random solutions.

2- Thirty different fully-correct seeds, generated in a previous run of 5-objective IBEA, along with 270 random solutions.

3- One "low-feature" seed generated using Z3 SMT solver, along with 299 random solutions.

TABLE IV. RESULTS FOR IBEA AND NSGA-II WITH AND WITHOUT FEATURE FIXING, 5 OPTIMIZATION OBJECTIVES

Model	NSGA-II without feature fixing			NSGA-II with feature fixing			IBEA without feature fixing			IBEA with feature fixing		
	%Correct	*TT50%*	*HV*	*%Correct*	*TT50%*	*HV*	*%Correct*	*TT50%*	*HV*	*%Correct*	*TT50%*	*HV*
Toybox	0.67%	N/A	0.14	12.5%	N/A	0.21	2.8%	N/A	0.25	**25%**	N/A	0.22
axTLS	0.67%	N/A	0.10	3.3%	N/A	0.21	4.7%	N/A	0.25	**100%**	157	0.21
eCos	1.33%	N/A	0.074	2%	N/A	0.082	**100%**	183	0.32	100%	113	0.33
FreeBSD	0.17%	N/A	0.001	0.5%	N/A	0.024	**91%**	688	0.32	**98%**	502	0.34
Fiasco	0.67%	N/A	0.084	2%	N/A	0.18	2.7%	N/A	0.23	**100%**	585	0.20
uClinux	0.67%	N/A	0	3.3%	N/A	0.16	1.5%	N/A	0	**31%**	N/A	0.30
Linux	0%	N/A	0	0%	N/A	0	0%	N/A	0	0%	N/A	0.021

[a] *%Correct: Percentage of correct solutions. TT50%: time to achieve 50% correctness (in seconds). HV: Hypervolume.*

[b] *Each cell reports the median value for 10 independent runs, each run for 30 minutes.*

TABLE V. GENERATING A CORRECT SOLUTION USING 2 METHODS

Model	Total Features	Using Z3		Using 2-obj IBEA	
		Time (sec)	*Selected Features*	*Time (sec)*	*Selected Features*
ToyBox	544	0.06	34	10.5	145
axTLS	684	0.06	81	16.5	245
eCos	1244	0.06	**0**	56	967
FreeBSD	1396	0.28	5	205	946
Fiasco	1638	0.07	248	42	575
uClinux	1850	0.01	7	23	455
Linux X86	6888	1.22	130	11,000 (~3 hours)	5704

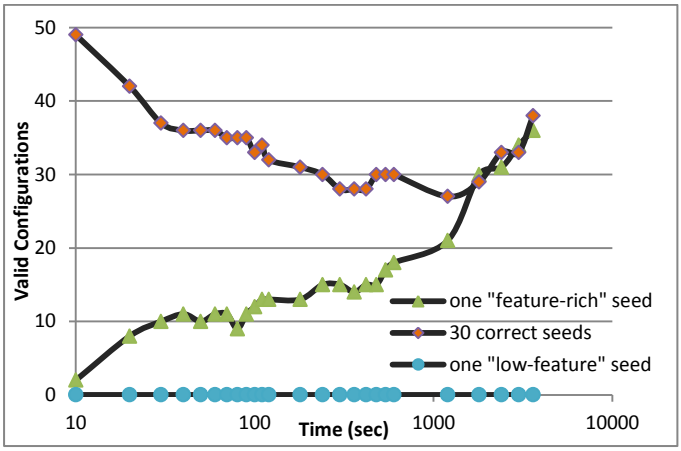

Fig. 6. Number of valid configurations over time, IBEA, 5-objectives with seeding, Linux X86 feature model

Figure 6 shows the growth of the number of correct solutions over time for all three types of seeds. The results for the one "feature-rich" show that 30 correct solutions are available after 30 minutes, and 36 such solutions are found after 1 hour. Compare this to the result in Table IV, which shows zero correct solutions for all 10 runs of 30 minutes each.

For the 30 seeds planted along with 270 random solutions, the population had 49 fully-correct configurations at 10 seconds, but the number dropped as the 5-objective optimization continued, down to 29 valid solutions at 30 minutes, and then back up to 38 after 1 hour. This shows that the outcome of 1 carefully-selected feature-rich seed is compatible with that of 30 seeds. *The quality, not quantity, of seeds had the most influence in the ability to scale up IBEA's optimization ability to the 6888-feature Linux model.*

For the "low-feature" seed obtained with Z3 SMT solver, the result was disappointing; no correct solutions were found for the first hour. We attribute this failure to the scarcity of selected features in the Z3 solution (130 features only). A solution with so small number of 1's would easily be "overrun" by other incorrect individuals in the population through the crossover process. In addition, this correct individual is assigned a low fitness value due to the low number of features, which decreases its likelihood of survival. The "feature-rich" seed obtained with 2-objective IBEA had 5704 selected features and thus was able to influence other individuals and stay ahead in the fitness evaluation.

VI. DISCUSSION

A. Method Innovation is Key to Scalability

A traditional view about the scalability of evolutionary algorithms is that the technology (i.e. CPU power, RAM) needs to catch up with the algorithms, since the population-based evolutionary methods require large amounts of RAM to store the primary population and the archive resulting from crossover and mutation, and CPU power would help finish the computations within reasonable time. Multicore CPUs would allow for the parallelization of execution, which is an important property of population-based methods. [11]

Our experience, as reported in this paper, was that large memory and fast CPUs were not enough to handle the size and complexity of the very large Linux model (6888 features). It took hours for the 5-objective optimization process to find any valid configurations, and more hours to find a significant set of valid solutions that are closer to optimality.

The innovation in method –the "seeding" technique– was our key to scalability. One feature-rich valid seed in the midst of a 300-member initial population was enough to generate 30 valid configurations within 30 minutes. A larger set of seeds did not help in improving the result, which hinted that the careful selection of seeds was more effective than increasing their quantity. One effective seed acted like the proverbial "straw that broke the camel's back".

B. Impact of the Scale-Up Result on Interactive Configuration

Configuration of a software package is an interactive process during which the users would make initial choices and then seek advice from the optimizer, and then make more choices, and so forth. The user's choices can be in the decision space (e.g. select an optional feature, select an option from a group), or they can be in the objective space (e.g. specifying range for cost, maximum acceptable risk). The more choices the user makes, the less complex the search space becomes, and the faster the optimizer can respond. The advantage of Pareto-optimal solutions is that they offer a range of options, rather than a unique optimal solution. Thus the user would be more informed and enabled in the configuration process.

The breakthrough that we achieved in this paper, via the seeding technique, enables jump-starting the configuration of 6000+ features by offering 10 valid options within the first minute (see the triangles in Fig. 6). Those 10 options are not just valid, but they "dominate" a host of other candidates in the Pareto sense, although they don't represent the absolute optimal Pareto front. The user can choose to begin making configuration decisions that early in the process. The optimizer takes the user's input, which narrows down the search space, and builds on the candidates achieved so far, and turns around with more good candidates that cater to the user's preferences. The seed, which is pre-computed offline, serves as an accelerator to the interactive configuration process.

C. Evolutionary Learning Still Rules

Without the remarkable result we presented in this paper, the slow convergence toward correct solutions may tempt us to abandon MEOAs and go directly to theorem provers, find all

possible product variants and evaluate them all. Such approach may be feasible with small and simple SPLs, but would not be scalable to large and complex ones, such as the Linux feature model, and the run times become prohibitive. Evolutionary methods (and especially IBEA) are still the best way to optimize with many objectives and vast decision spaces. Navigating the decision space with the aid of heuristics and simultaneously evaluating a population of candidates has proven to outperform exhaustive search over many years of research. This trend should continue to scale up with the help of innovative techniques that inhibit the randomness of exploration and nudge the optimizer towards respecting domain constraints. The seeding trick is one such helper.

D. Be Careful with Problem Formulation

Many-objective optimization is a paradigm shift that forces researchers to reformulate traditional problems in order to bring out the various objectives and map out a Pareto front. The same problem can have different formulations according to which parameters the researcher chooses to bring out as independent dimensions. In a recent survey [21], we found that most researchers only examined two-objective formulations of their problems. But we also found several examples of the same researchers addressing different formulations of the same problems by varying the number of objectives.

The most interesting problem formulation would have a maximum number of objectives among which there are minimal correlations. Such formulation would challenge the multiobjective optimizer to find the set of best trade-offs among competing objectives. On the other hand, if the formulation separates two objectives among which there's a high correlation, then the optimization takes a monolithic direction, with the solutions crowding in the same area that tends to optimize both objectives at the same time.

A case in point is our 5-objective formulation of the software configuration problem. We seek to 1) minimize violations, 2) maximize features, 3) minimize newly-developed features (not used before), 4) minimize known defects, and 5) minimize cost. Some have looked at the second objective and questioned its merit; does the user really seek to maximize the number of features in a product? Our answer takes a holistic look at the goal of our optimization: to provide the user with a wide range of Pareto-optimal solutions that explore as many feature configuration choices as possible, and then let the user make their own decisions. If the "feature-richness" objective is removed, the other four objectives would push the solutions toward minimizing the number of features, since that area of the decision space tends to decrease violations, new features, known defects, and cost. Such formulation would defeat the overall purpose of offering a diverse set of valid configurations.

E. Confirming IBEA's Advantage in Many-Objective Problems

The results of this experiment confirm the findings of earlier work by the authors [23] [22] and by Wagner et al. [27] regarding the superiority of IBEA over other Pareto-based algorithms (such as NSGA-II) in high-dimensional objective spaces. This is attributed to IBEA's fitness assignment strategy which heavily factors in the user preferences, whereas Pareto-based methods rely on absolute dominance as primary fitness criterion and diversity as secondary criterion, which tend to ignore differences in quality that IBEA is able to capture.

F. Building on the Seeding Approach

In light of the limited success of the seeding technique, we suggest the use of a pre-computed set of seeds (a set of correct solutions), given that it is diverse in the amount of selected features. The more selected features the better chance there will be of promoting correct solutions in the many-objective optimization problem. Furthermore, since the Z3 SMT solver can arrive at valid solutions much faster than the 2-objective IBEA, we will try to create the desired set of seeds using Z3.

VII. Threats to Validity

In the first part of the results, i.e. the feature fixing technique, we repeated each algorithm run 10 times for each of the 7 models. We performed the Mann-Whitney test and found significant improvements in the %Correct indicator for 6 out of 7 models. This should be sufficient to eliminate a potential threat to conclusion validity.

As for the second part of the results, i.e. the seeding technique, we didn't validate the findings with the same level of repeats and statistical testing. We plan to do so in future work as we explore the proper characterization of the effective seeds and reduce the time needed to generate them.

A potential threat to construct validity is the use of synthetic data as attributes of features, i.e. COST, DEFECTS, and USED_BEFORE. The use of synthetic data is common in software engineering literature. The difficulty of obtaining real data comes from the fact that such data are usually associated with software components, not features. When available, such data is often proprietary and not published. Nevertheless, the results we obtained have such a large margin of superiority achieved by IBEA with feature fixing over other methods which couldn't possibly be biased by the synthetic data.

VIII. Conclusion and Future Work

This experiment explored the scalability of optimum 5-objective product configuration using IBEA for very large feature models. For models with less than 2000 features, IBEA was able to achieve the goal within reasonable time only with the help of feature fixing. As for the Linux kernel, a 6888-feature model, we were able to achieve 30 valid configurations within 30 minutes with the help of an innovative population-seeding technique. One pre-computed feature-rich solution was enough to influence the rest of the population into learning valid solutions faster.

Future work will focus on characterizing the quality and quantity of the best seed which would be most influential in helping IBEA to converge faster to large amounts of valid solutions. In addition, the Z3 SMT solver will be utilized to generate the desired set of seeds in shorter times than possible using evolutionary methods.

Acknowledgment

This research work was funded by the Qatar National Research Fund (QNRF) under the National Priorities Research

Program (NPRP) Grant No.: 09-1205-2-470. Additional funding by the National Science Foundation (NSF) Grant No.: CCF 1017330.

REFERENCES

[1] D. Benavides, A. Ruiz-Cortés, and P. Trinidad, "Automated Reasoning on Feature Models," in *Proc. CAISE*, 2005, pp. 491-503.

[2] D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," *Information Systems*, vol. 35, no. 6, pp. 615-636, 2010.

[3] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "Feature-to-Code Mapping in Two Large Product Lines," Department of Computer Science, University of Leipzig, Leipzig, Technical Report 2010. [Online]. http://gsd.uwaterloo.ca/sites/default/files/2010-TR-presence_conditions.pdf

[4] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "Variability Modeling in the Real: A Perspective from the Operating Systems Domain," in *Proc. ASE*, Antwerp, Belgium, 2010, pp. 73-82.

[5] 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, Waterloo, Canada, Technical Report GSDLAB–TR 2012–07–06, 2012. [Online]. http://gsd.uwaterloo.ca/sites/default/files/vm-2012-berger.pdf

[6] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proc. TACAS, LNCS 4963* , Budapest, Hungary, 2008, pp. 337-340.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.

[8] J.J. Durillo and A.J. Nebro, "jMetal: A Java Framework for Multi-Objective Optimization," *Advances in Engineering Software*, vol. 42, pp. 760-771, 2011.

[9] G. Fraser and A. Arcuri, "The Seed is Strong: Seeding Strategies in Search-Based Software Testing," in *Proc. ICST*, 2012, pp. 121-130.

[10] 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," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2208–2221, December 2011.

[11] M. Harman, "Software Engineering Meets Evolutionary Computation," *IEEE Computer*, vol. 44, no. 10, pp. 31-39, October 2011.

[12] M. Harman, S.A. Mansouri, and Y. Zhang, "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications," King's College, London, UK, Technical Report TR-09-03, 2009.

[13] C. Henard et al., "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines ," *arXiv preprint*, pp. arXiv:1211.5451, 2012.

[14] M.F. Johansen, O. Haugen, and F. Fleurey, "Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible," in *Proc. MODELS'11, LNCS 6981*, 2011, pp. 638-652.

[15] S.Q. Lau, "Domain Analysis of E-Commerce Systems using Feature-Based Model Templates," Dept. Electrical and Computer Engineering, University of Waterloo, Canada, Master's Thesis 2006.

[16] R.E. Lopez-Herrejon and A. Egyed, "Searching the Variability Space to Fix Model Inconsistencies: A Preliminary Assessment," in *Proc. SSBSE*, Szeged, Hungary, 2011.

[17] M. Mendonca, M. Branco, and D. Cowan, "S.P.L.O.T. - Software Product Lines Online Tools," in *Proc. OOPSLA*, Orlando, USA, 2009.

[18] M. Mendonca, A. Wąsowski, and K. Czarnecki, "SAT-Based Analysis of Feature Models is Easy," in *Proc. SPLC*, San Francisco, USA, 2009.

[19] M. Mendonca, A. Wąsowski, K. Czarnecki, and D. Cowan, "Efficient Compilation Techniques for Large Scale Feature Models," in *Proc. GPCE*, Nashville, USA, 2008.

[20] R. Pohl, K. Lauenroth, and K. Pohl, "A Performance Comparison of Contemporary Algorithmic Approaches for Automated Analysis Operations on Feature Models," in *Proc. ASE*, Lawrence, KS, USA, 2011, pp. 313-322.

[21] A.S. Sayyad and H. Ammar, "Pareto-Optimal Search-Based Software Engineering: A Literature Survey," in *Proc. RAISE*, San Francisco, USA, 2013.

[22] 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 *Proc. CMSBSE*, San Francisco, USA, 2013.

[23] 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 *Proc. ICSE*, San Francisco, USA, 2013, pp. 492-501.

[24] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki, "Reverse Engineering Feature Models," in *Proc. ICSE'11*, Honolulu, USA, 2011.

[25] R. Shi, J. Guo, and Y. Wang, "A Preliminary Experimental Study on Optimal Feature Selection for Product Derivation using Knapsack Approximation," in *Proc. PIC*, 2010, pp. 665-669.

[26] S. Soltani, M. Asadi, H. Marek, D. Gasevic, and E. Bagheri, "Automated Planning for Feature Model Configuration based on Stakeholder's Business Concerns," in *Proc. ASE*, Lawrence, KS, USA, 2011, pp. 536-539.

[27] T. Wagner, N. Beume, and B. Naujoks, "Pareto-, Aggregation-, and Indicator-Based Methods in Many-Objective Optimization," in *Proc. EMO, LNCS Volume 4403/2007*, 2007, pp. 742-756.

[28] 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, August 2009.

[29] J. White, B. Dougherty, D.C. Schmidt, and D. Benavides, "Automated Reasoning for Multi-Step Feature Model Configuration Problems," in *Proc. SPLC*, San Francisco, USA, 2009, pp. 11-20.

[30] J. White, D.C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz–Cortés, "Automated Diagnosis of Product-line Configuration Errors in Feature Models," in *Proc. SPLC*, 2008, pp. 225-234.

[31] J. White, D.C. Schmidt, A. Nechypurenko, and E. Wuchner, "Optimizing and Automating Product-Line Variant Selection for Mobile Devices," in *Proc. MOBISYS*, Puerto Rico, 2007.

[32] G. Zhang, H. Ye, and Y. Lin, "Using Knowledge-Based Systems to Manage Quality Attributes in Software Product Lines," in *Proc. SPLC*, 2011.

[33] E. Zitzler and S. Kunzli, "Indicator-Based Selection in Multiobjective Search," in *Parallel Problem Solving from Nature*. Berlin, Germany: Springer-Verlag, 2004, pp. 832–842.

[34] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.