2014

Evolutionary Search Techniques with Strong Heuristics for Multi-Objective Feature Selection in Software Product Lines

Abdel Salam Sayyad
West Virginia University

Evolutionary Search Techniques with Strong Heuristics for Multi-Objective Feature Selection in Software Product Lines

by
Abdel Salam Sayyad

Dissertation Submitted to
the Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
In
Computer Engineering

Approved by:

Dr. Hany Ammar, Committee Chair
Dr. Tim Menzies, Committee co-Chair
Dr. Ramana Reddy
Dr. Ali Mili
Dr. Mario Perhinschi

Lane Department of Computer Science and Electrical Engineering

Morgantown, WV, USA
August 2014

Keywords: Software Product Lines, Metaheuristic Search Algorithms, Multi-Objective Optimization, Search-Based Software Engineering

ABSTRACT

Evolutionary Search Techniques with Strong Heuristics for Multi-Objective Feature Selection in Software Product Lines

Abdel Salam Sayyad

Software design is a process of trading off competing objectives. If the user objective space is rich, then we should use optimizers that can fully exploit that richness. For example, this study configures software product lines (expressed as feature models) using various search-based software engineering methods. Our main result is that as we increase the number of optimization objectives, the methods in widespread use (e.g. NSGA-II, SPEA2) perform much worse than IBEA (Indicator-Based Evolutionary Algorithm). IBEA works best since it makes most use of user preference knowledge. Hence it does better on the standard measures (hypervolume and spread) but it also generates far more products with 0 violations of domain constraints. We also present significant improvements to IBEA's performance by employing three strong heuristic techniques that we call PUSH, PULL, and seeding. The PUSH technique forces the evolutionary search to respect certain rules and dependencies defined by the feature models, while the PULL technique gives higher weight to constraint satisfaction as an optimization objective and thus achieves a higher percentage of fully-compliant configurations within shorter runtimes. The seeding technique helps in guiding very large feature models to correct configurations very early in the optimization process. Our conclusion is that the methods we apply in search-based software engineering need to be carefully chosen, particularly when studying complex decision spaces with many optimization objectives. Also, we conclude that search methods must be customized to fit the problem at hand. Specifically, the evolutionary search must respect domain constraints.

Dedication

To my family, who supported me and endured with me throughout this journey.

Acknowledgments

My deepest thanks and appreciation go to my PhD advisors: Prof. Hany Ammar and Prof. Tim Menzies, for their immense guidance and endless support.

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, and the National Science Foundation (NSF) Grant No.: CCF 1017330.

Table of Contents

List of Tables

List of Figures

List of Acronyms

BDD	Binary Decision Diagram
CSP	Constraint Satisfaction Problem
CTC	Cross-Tree Constraint
DIMACS	Discrete Mathematics and Theoretical Computer Science
FastPGA	Fast Pareto Genetic Algorithm
HV	Hypervolume
IBEA	Indicator-Based Evolutionary Algorithm
LVAT	Linux Variability Analysis Tools
MOCell	Multi-Objective Cellular Genetic Algorithm
MOEA	Multi-Objective Evolutionary Algorithm
NSGA	Nondominated Sorting Genetic Algorithm
POSBSE	Pareto-Optimal Search-Based Software Engineering
SAT	The Satisfiability Problem
SBSE	Search-Based Software Engineering
SMT	Satisfiability Modulo Theories
SPEA	Strength Pareto Evolutionary Algorithm
SPL	Software Product Line
SPLE	Software Product Line Engineering
SPLOT	Software Product Line Online Tools

Chapter 1: Introduction to the Study

Software engineering technologies are increasingly having a direct impact on business outcomes, so much so that software decisions must be value-driven early on, i.e. business concerns ought to play a central role in the selection of software tools, technologies and processes. Barry Boehm states that: "software has a major influence on most systems' cost, schedule, and value; and value-neutral software decisions can seriously degrade project outcomes." [**16**] This comment is most relevant to the subject of this thesis, in which we choose our evolutionary optimization algorithm to best exploit the user preferences in search for optimized feature selections in a software product line (SPL). This problem entails searching the space of possible feature configurations for sets of feature selections that conform to domain constraints, but another goal is to increasingly optimize the objectives that the user cares about, such as cost and reliability metrics.

The global information technology infrastructure is becoming increasingly reliant on increasingly complex software. This study is about the way to rapidly explore and configure such complexity. Feature models allow visualization, reasoning, and configuration of large and complex software product lines (SPLs). Common SPLs now consist of hundreds (even thousands) of features, with complex dependencies and constraints that govern which features can or cannot live and interact with other features. For instance, according to **[**14**]**, the Linux model has 6320 features, of which 86% declare

constraints of some sort, and most features refer to 2-4 other features. Such level of complexity surely requires automated reasoning and configuration techniques, especially if the intricacies of the feature model are combined with further user preferences and priorities, such as those related to cost and reliability. In that case, the job of offering product variants with guaranteed conformance to the feature model and efficiency according to user preferences becomes monumental, requiring tool assistance.

Many Search-Based Software Engineering (SBSE) researchers who seek to apply metaheuristic search methods to their problems choose the algorithms based on popularity of the algorithm in the SE literature, or its wide usage in other fields. Harman makes the following comment regarding the choice of evolutionary search algorithms over non-evolutionary ones: "We must be wary of the unquestioning adoption of evolutionary algorithms merely because they are popular and widely applicable or because, historically, other researchers have adopted them for SBSE problems; none of these are scientific motivations for adoption." [46] We believe the same comment applies to choosing certain evolutionary algorithms over others.

Most importantly, the chosen algorithms have to accommodate both the user preferences and the structure of the decision space. In [86] and [85], we use IBEA (Indicator-Based Evolutionary Algorithm) to best exploit user preference knowledge, but the search takes hours to produce a significant amount of model-conforming solutions. In more recent results, we quickly achieve 100% fully-conformant solutions, and the remaining time is spent on further optimizing solutions according to user preferences. We

accomplish this remarkable gain by introducing three strong heuristic techniques: PUSH, PULL, and population seeding. The PUSH technique first learns certain rules from static analysis of the feature model, and then forces the evolutionary process to respect those rules. The PULL technique gives more weight to constraint satisfaction in the optimization process, and thus achieves full conformance in all solutions within shorter time. The seeding technique pre-computes one valid configuration and then implants it in the initial population as a model for the other individuals to follow, which helps find many good solutions sooner in many-objective optimization of feature models with thousands of features and hundreds of thousands of constraints. We demonstrate the usefulness of our methods by optimizing the configuration of 20 medium-sized academic feature models from the SPLOT repository (Software Product Line Online Tools)[1], in addition to 7 large real feature models from the LVAT repository (Linux Variability Analysis Tools)[2].

The scalability of our results to large feature models derived from actual software systems (LVAT) is of significant importance. Scalability of SBSE methods 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. **[**50**]**. They

[1] http://www.splot-research.org
[2] https://code.google.com/p/linux-variability-analysis-tools/source/browse/?repo=formulas

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 [**68**]. Examples are: Pohl et al. [**75**], Lopez-Herrejon and Egyed [**64**], Johansen et al. [**55**], Mendonca et al. [**70**] and our own previous work [**86**], [**84**]. 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 [**62**], 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. [**15**] 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. 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. [**55**] who generated test covering arrays for feature models, and Henard et al. [**54**] 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. [**105**] [**104**] [**103**] [**106**], Shi et al. [**88**], Guo et al. [**45**], and Mendonca et al. [**70**]. 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.

1.1 Contributions of This Work

The overall contributions of our work in optimum feature selection in software product lines are summarized as follows:

1- The formulation of this problem as a multi-objective optimization problem which requires a Pareto-efficient set of solutions. Although related work (see section 3.2) also defined secondary objectives in feature model configuration, they aggregated the multiple objectives into a single fitness function, and searched for a single optimum solution, which is of less benefit to the end user.

2- We use Multi-Objective Evolutionary Algorithms (MOEAs) for the first time to provide solutions for this problem, and comparing the performance of multiple MOEAs based on quality indicators and the amount of solutions that conform to the feature models. We demonstrate the remarkable advantage of the Indicator-Bases

Evolutionary Algorithm (IBEA). Wagner et al. [**98**] show that Indicator-Based algorithms (such as IBEA) perform remarkably better than Pareto-Based algorithms (such as NSGA-II) when applied to many-objective problems. Our research was the first to confirm that result in software engineering.

3- We validate our results using 20 medium-sized feature models developed as academic examples and 7 large feature models that were reverse-engineered from actual projects. The total number of features ranged between 43 and 6888 features.

4- We achieve fast run times which allow the practical use of IBEA in interactive configuration of feature models. Within 1 second, we are able to provide the user with a wide range of model-conforming solutions. This is possible with the help of the PUSH and PULL techniques, and with better-tuned evolutionary parameters.

5- A breakthrough scalability result. Using the population seeding technique we now show that it is possible to configure product lines as large as 6000+ features, and 344,000 constraints.

1.2 Published Results

In our earliest work [**86**] we tackled the problem of configuring software product lines as a high-dimensional multi-objective search problem, and we compared Pareto-optimal solutions from several algorithms to show the superiority of IBEA. In that paper, we only used two feature models, and we experienced long run times which can only correspond with offline optimization.

In [**84**] and [**83**], we improved the performance of IBEA with regard to configuration of feature models by reducing the rates of crossover and mutation. We found that the "rule-of-thumb" rates often used with evolutionary algorithms amount to lengthening the duration of the search. This highlights the fine-grained structure of the feature models, where small changes in features have great effects on other features, and thus the search needs to proceed slowly for better exploration of the decision space.

In [**85**], we compare the performance of IBEA against NSGA-II when applied to large feature models from the LVAT (Linux Tools) repository. We utilize "feature fixing" as a way to force the search to respect certain rules that we learn from the feature models. We also report on an early result using the "population seeding" heuristic, which we expand in this document.

1.3 Recent Results

Here, we introduce the "Tree Mutation" operator which only mutates (flips) a feature selection decision if that mutation is allowed by the tree structure, i.e. the mutation is not allowed to override parent-child dependencies or group selection restrictions. Tree mutation is only applied to SPLOT models since they explicitly represent the feature tree. For LVAT models, we employ "feature fixing" which was introduced in [**85**]. Together, "tree mutation" and "feature fixing" are called the PUSH technique in this paper; since they both restrict the evolutionary process using pre-learned knowledge about feature model structure.

We also introduce the PULL technique, which give more weight to constraint violation as a minimization objective; thus achieving significant amounts of model-conforming configurations sooner and accelerating the multi-objective search. Together, the PUSH and PULL techniques achieve runtime improvement of over 16,000 times for the E-Shopping feature model than achieved in [**86**].

We present the results of an extensive experiment comparing five different algorithms applied to 27 software product lines. IBEA with PUSH and PULL emerged, as expected, as the ultimate optimization tool with regard to the configuration of hierarchically-modeled software systems.

1.4 Statement of Thesis

We propose that the optimizers we choose must be considered according to the nature of the problem at hand; for example, when the user preference space is rich, choose an optimizer that exploits that richness. In particular, we find that an optimizer which uses continuous dominance for its fitness raking is better at many-objective optimization than those depending on Boolean dominance.

In addition, we advocate the customization of our optimizers to the problem domain such that it searches for solutions while respecting domain constraints. This calls for the deployment of strong heuristics which guide the search algorithm into promoting solutions that are close to domain conformance; rather than spending search and evaluation time on less conforming solutions.

1.5 Organization of This Document

The rest of this document is organized as follows: Chapter 2 provides background material on software product lines, genetic algorithms, and MOEAs. Chapter 3 introduces the problem formulation. Chapter 4 reviews related work in the use of MOEAs in software engineering. Chapter 5 describes the experimental setup, and chapter 6 presents the experimental results. Chapter 7 presents the threats to validity. In chapter 8, we discuss our findings and their implications. Finally, chapter 9 presents conclusions and future work.

Chapter 2: Background

2.1 Software Product Line Engineering (SPLE)

Increasingly, software engineers spend their time creating software families consisting of similar systems with many variations [**26**]. Many researchers in industry and academia started using a feature-oriented approach to commonality and variability analysis after the Software Engineering Institute introduced Feature-Oriented Domain Analysis (FODA) in 1990 [**56**]. Software product line engineering is a paradigm to develop software applications using platforms and mass customization. Benefits of SPLE include reduction of development costs, enhancement of quality, reduction of time to market, reduction of maintenance effort, coping with evolution and complexity [**74**], and identifying opportunities for automating the creation of family members [**26**].

2.2 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 1, adapted from [**11**], depicts a simplified feature model inspired by the mobile phone industry.

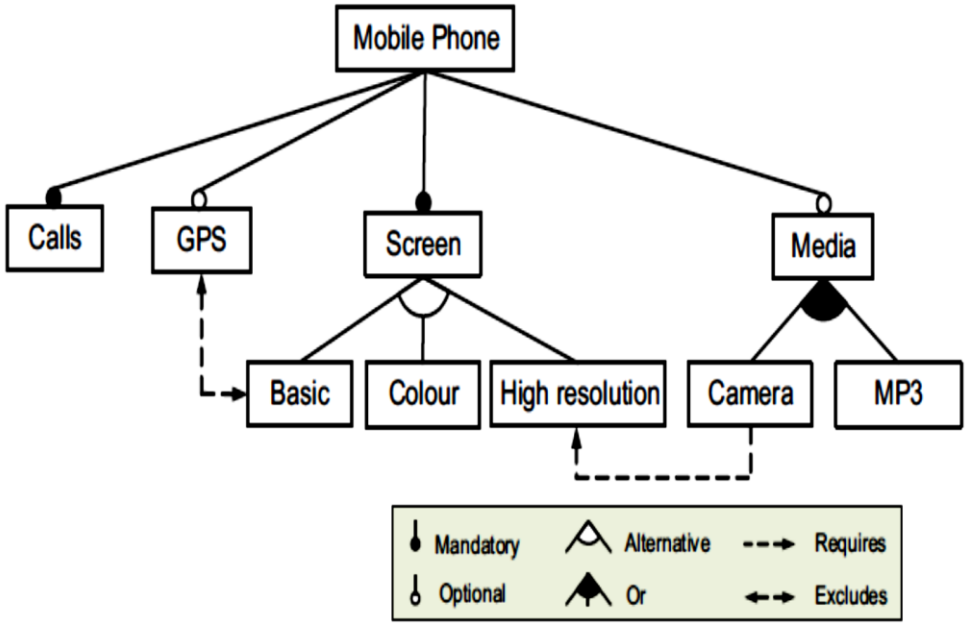

Figure 1: Feature model for mobile phone product line, adopted from [3]

The Simple XML Feature Model (SXFM) format was defined by the SPLOT website [**68**], which was launched in May 2009. SPLOT is host to a feature model repository which adheres to the SXFM format. Figure 2 shows the SXFM format for the mobile phone feature model from Figure 1. It shows the root feature (marked with :r), the mandatory features (marked with :m), the optional features (marked with :o), and the group features (marked with :g). The cross-tree constraints are listed at the bottom in Conjunctive Normal Form (CNF).

```
<feature_model name="Mobile-Phone">
<meta>
<data name="description">Mobile Phone</data> …
</meta>
<feature_tree>
:r Mobile Phone(root)
        :m calls(calls)
        :o GPS(gps)
        :m screen(screen)
                :g (_g_1) [1,1]
                        : basic(basic)
                        : color(color)
                        : high-resolution(hi_res)
        :o media(media)
                :g (_g_2) [1,*]
                        : camera(camera)
                        : mp3(mp3)
</feature_tree>
<constraints>
constraint_1: ~gps or ~basic
constraint_2: camera or ~hi_res
</constraints>
</feature_model>
```

Figure 2: Mobile phone feature model in SXFM format

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)

Figure 3: Mobile phone feature model as a Boolean expression

The full set of rules in a feature model can be captured in a Boolean expression. In [**8**] and [**29**], a procedure is provided for converting feature models into propositional formulas, where the features are treated as propositional variables, and the relationships are expressed in terms of conjunction ∧, disjunction ∨, negation ¬, implication →, and bi-implication ↔. Figure 3 above shows the mobile phone feature model of Figure 1 converted into a propositional formula. The last two lines represent the cross-tree constraints (CTCs) in the model.

From Figure 3 we can conclude that the total number of rules in this feature model is 16, including the following:

1. The root feature is mandatory.
2. Every child requires its own parent.
3. If the child is mandatory, then the parent requires the child.
4. Every group adds a rule about how many members can be chosen.

5. Every cross-tree constraint (CTC) is a rule.

The total number of rules will be used as the "full correctness" score in this study, thus making "correctness" one of the optimization objectives. (see section 3.4)

Another source of feature models used in this study was the Linux Variability Analysis Tools (LVAT) feature model repository, which resulted from the works of Berger et al. [**15**], [**87**], [**14**], [**13**], [**12**]. 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 [**68**]. 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 models downloaded from LVAT website had the DIMACS format, which expresses each model as a formula in the Conjunctive Normal Form (CNF).

2.3 Tools for Automated Analysis of Feature Models

Here we describe the conventional formula solvers (a.k.a. theorem provers) that are traditionally used for analyzing feature models. Those solvers include BDD (Binary Decision Diagram) solvers, SAT (Satisfiability) solvers, and CSP (Constraint Satisfaction Problem) solvers. We also discuss an SMT (Satisfiability Modulo Theories) solver that was more recently used for the same purpose.

2.3.1 Binary Decision Diagrams (BDD)

Binary Decision Diagrams are compact data structures for Boolean expressions widely studied in the hardware verification community [**80**]. A BDD is a directed acyclic graph that represents a Boolean expression.

For example, the expression

$(k \rightarrow p) \wedge (p \rightarrow c)$

is represented by the BDD in Figure 4. [**29**]

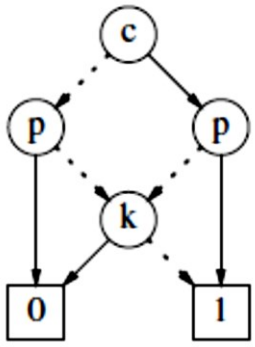

Figure 4: Example Binary Decision Diagram

Using a BDD, the Boolean expression can efficiently be evaluated to find valid solutions. For instance, it is possible to test in constant time whether a BDD is constantly true or false, although for Boolean expressions this problem is NP-complete [**1**].

2.3.2 The Satisfiability (SAT) Problem

The problem of determining the Satisfiability of sentences in propositional logic is called the SAT problem [**80**]. It was the first problem proved to be NP-complete [**25**]. Boolean Satisfiability is a decision problem to check whether a Boolean formula

evaluates to true for any of the assignments to its variables. If there is such assignment the formula is said to be satisfiable, otherwise it is unsatisfiable. For instance, (a ∧ b) is satisfiable as witnessed by the assignment {a=true, b=true}. Formula (a ∧ b ∧ ¬a) is unsatisfiable: no value for variable a causes the formula to evaluate to true [**75**].

2.3.3 The Constraint Satisfaction Problem (CSP)

The constraint satisfaction problem includes a set of variables, each having a range of allowable values. A problem is solved when each variable has a value that satisfies all the constraints on the variable. CSP search algorithms enable the solution of complex problems by eliminating large portions of the search space all at once by identifying variable/value combinations that violate the constraints. [**80**] A CSP – in contrary to a SAT– includes constraints containing integer and interval values [**69**]. The algorithm to transform a feature model into a CSP was introduced in [**10**].

2.3.4 Satisfiability Modulo Theories (SMT)

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. [**31**]

2.3.5 The Infeasibility of Exhaustive Search for Optimal Configuration

All the methods discussed above (BDD, SAT, CSP, and SMT) are concerned with finding correct solutions to logic expressions in the shortest time possible. Nevertheless, when searching for optimal configuration options with multiple quality attributes (as in the problem addressed by this study, see section 3.1), it is required that all correct solutions be found and evaluated, which becomes an infeasible task as feature models grow in size and complexity.

For example, Pohl et al. [**75**], who experimented with finding solutions for feature models using BDD, SAT, and CSP solvers, ruled that the task of finding all solutions for a feature model becomes infeasible when the cardinal (number of valid solutions) is over 3 million, hence excluding feature models as small as 90 features in size.

Olaechea Velazco [**96**] used Z3 SMT solver to find optimal solutions to the two feature models and quality attributes that we used in [**86**]. For the smaller model with 43 features only, the process took 1.65 hours. As for the larger model, with 290 features, the process timed out and was stopped after 15 days. The cardinal for this 290-feature model is 2.26×10^{49} [**75**].

Other examples that show the hardness of exhaustive search for large feature models are mentioned in section 4.1.

Hence we resorted to heuristic methods for this task, namely multi-objective evolutionary algorithms (MOEAs) that are explained in the following sections.

2.4 Genetic Algorithm

All MOEA's used in this study have Genetic Algorithm at their core. The original Genetic Algorithm (GA) was developed with a single fitness value in mind, i.e. a single optimization objective. Figure 5 lists an outline of the Genetic Algorithm, adapted from [**65**].

The **MatingSelection**() function selects individuals from the population with a probability proportionate with their fitness. An individual may, by chance, be chosen for breeding multiple times.

```
Initialization
        popsize ← desired population size
        P ← {}
        for popsize times do
                P ← P ∪ {new random individual}
        Best ← null
Repeat until Termination
      for each individual Pi ∈ P do
        AssignFitness(Pi)
        if Best = null or Fitness(Pi) > Fitness(Best) then
              Best ← Pi
      Q←{}
      for popsize/2 times do
        Parent Pa ← MatingSelection(P)
        Parent Pb ← MatingSelection (P)
        Children Ca,Cb ← Crossover(Copy(Pa),Copy(Pb))
        Q ← Q ∪{Mutate(Ca),Mutate(Cb)}
      P ← Q
Termination Best is ideal solution or we've run out of
time
return Best
```

Figure 5: Genetic Algorithm, adapted from [22]

The individual solutions are of the Boolean string type. The **Crossover**() function performs a single-point crossover, where a point is chosen randomly, and the bits are swapped between the two parents as shown in Figure 6 [**65**]. Crossover between two parents is performed with a predetermined probability.

The **Mutate**() function performs bit-flip mutation, where each bit in the string may be flipped at a predetermined probability.

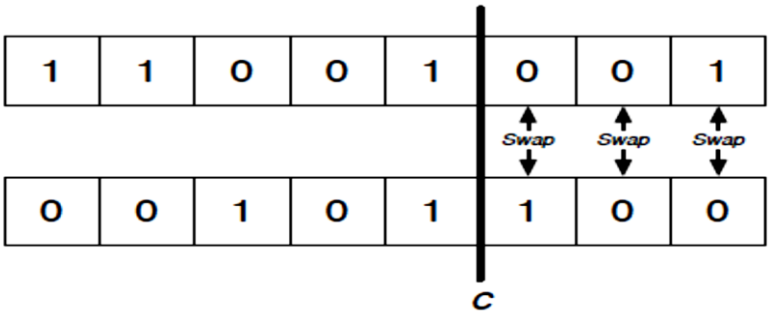

Figure 6: Example of a single-point crossover, from [22]

2.5 Multi-Objective Evolutionary Algorithms (MOEAs)

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 [**118**].

Formally, we define two types of Pareto Dominance: Boolean Dominance is defined as follows: 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.

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

whereas Continuous Dominance is measured with a continuous function, such as equation 2 in Figure 7.

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

Many algorithms have been suggested over the past two decades for multi-objective optimization based on evolutionary algorithms that were designed primarily for single-objective optimization, such as Genetic Algorithms, Evolutionary Strategies, Particle Swarm Optimization, and Differential Evolution.

The algorithms we used in this study were already implemented in the jMetal framework [**34**]. They are:

1. IBEA: Indicator-Based Evolutionary Algorithm [**116**].
2. NSGA-II: Nondominated Sorting Genetic Algorithm, version 2 [**33**].
3. SPEA2: Strength Pareto Evolutionary Algorithm, version 2 [**117**].
4. FastPGA: Fast Pareto Genetic Algorithm [**38**].
5. MOCell: A Cellular Genetic Algorithm for Multi-objective Optimization [**71**].

2.6 Indicator-Based Evolutionary Algorithm (IBEA)

Of the 5 algorithms listed above, IBEA achieved the best results, and that was due to its unique fitness ranking criteria. In Figure 7, we provide an outline of the IBEA algorithm. The details can be found in [**116**]. In the following section, we explain the

fitness ranking criteria in each of the 5 algorithms.

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

Step 1: **Initialize**: 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: **Assign Fitness**: Calculate fitness values of individuals in P, i.e., for all $\mathbf{x}_1 \in$ P set

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

Where $I_{HD}(.)$ is a dominance-preserving binary indicator. Specifically:

$$I_{HD}(x_2, x_1) = \begin{cases} I_H(x_1) - I_H(x_2) \; if \;\; x_2 dominates \; x_1 \\ I_H(x_2 + x_1) - I_H(x_2), \; otherwise \end{cases} \quad (3)$$

Where I_H is the Hypervolume defined in section 3.6 below.

Step 3: **Environmental selection**: Iterate the following three steps until the size of population P does not exceed α:
1. Choose an individual $\mathbf{x}*$ ∈ P with the smallest fitness value, i.e., $F(\mathbf{x}*) \leq F(\mathbf{x})$ for all $\mathbf{x} \in$ P.
2. Remove $\mathbf{x}*$ from the population.
3. Update the fitness values of the remaining individuals, i.e.

$F(x) = F(x) + e^{-I(\{x_2\},\{x_1\})/\kappa}$ for all $x \in P$.

Step 4: **Termination**: If m ≥ 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 temp mating pool P'.
Step 6: **Variation**: Apply **crossover** 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.

Figure 7: Indicator-Based Evolutionary Algorithm (IBEA)

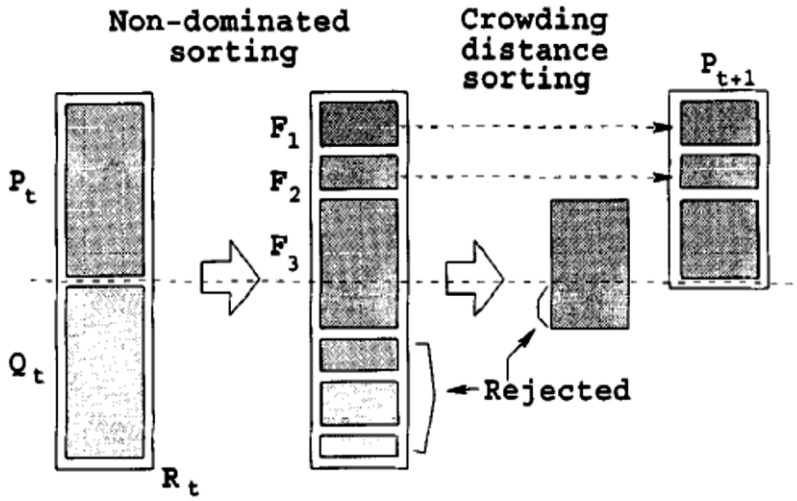

Figure 8: NSGA-II sorting procedure, from [13]

2.7 Fitness Ranking Criteria in MOEAs

All MOEAs used in this study are based on Genetic Algorithms. They share some basic qualities, such as: single-point crossover, bit-flip mutation, binary tournament for mating selection, and elitism. They also have differences, the most relevant to mention here being the ranking criterion (i.e. fitness assignment) used to determine which individuals have stronger chance to survive to the next generation. Those criteria are detailed here:

1. NSGA-II: The sorting procedure in NSGA-II is depicted in Figure 8, adopted from [**33**]. It shows how the combined primary population (P_t) and secondary population (Q_t) 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 to determine fitness for survival. [**33**]

2. SPEA2: The sorting procedure in SPEA2 is somewhat similar to that depicted in Figure 8, with two differences. First, domination sorting only takes place once, thus dividing the population into F1 and F2. Second, the ranking criterion for individuals in F2 is based on the strength of each solution, defined as the number of solutions that are dominated by it. The fitness value of a point is the sum of strengths of all solutions that dominate that point added to a density estimation that works to prioritize points with less proximity to nearest neighbors. [**117**]

3. FastPGA: The fitness of the nondominated solutions is calculated using the crowding distance as in NSGA-II. The fitness of dominated solution is the number of solutions it dominates, similar to SPEA2. Ranking is according to both domination and fitness. [**38**]

4. MOCell: The solutions are ordered using a ranking and a crowding distance estimator similar to NSGA-II. Bigger distance values are favored. [**71**]

5. IBEA: Equation 2 in Figure 7 shows IBEA's fitness assignment. Each solution is given a weight based on $I_{HD}(.)$, 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" [**116**]. 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 SPEA2, and followed by others.

2.8 Summary

In this chapter, we introduced all the background material necessary to move forward. In the majority of previous work on automated analysis of feature models, only one dimension is explored, namely the correctness dimension, such as checking the soundness of the model, and deriving correct configurations. There wasn't enough exploration of multiple dimensions that address user preferences such as cost, reliability, etc.

Going forward, we will examine feature models with added quality attributes that express stakeholder interests, and we will use MOEAs to find near optimal feature selection that conform with the dependencies and constraints laid out in the feature models.

Chapter 3: Problem Formulation

In this chapter, we define our problem, i.e. multi-objective configuration of software product lines, and we place it within the context of related work in reasoning about feature attributes and user preferences in feature models. In addition, we highlight the relationship of this problem with interactive or "user-in-the-loop" configuration.

3.1 Problem Definition

The problem of multi-objective feature selection in software product lines in formally defined as follows.

Suppose a feature model is composed of *n* features, denoted as $F = \{F_1, F_2, \ldots, F_n\}$, that the features are subject to a set of *m* constraints (including all dependencies, see Fig. 3), denoted as $C = \{C_1, C_2, \ldots, C_m\}$, a set of *k* user objectives $O_{Si} = \{O_{Si1}, O_{Si2}, \ldots, O_{Sik}\}$ where each objective is calculated for a given subset of features S_i, and a search budget of T seconds. Find a group of feature subsets $S_i \subset F$, such that:

a- $C_1 = C_2 = \ldots = C_m = \text{TRUE}$

b- O_{Si} is not dominated by any O_{Sj} found up to time T, $\forall$ i,j (domination defined in equation 1, section 2.5).

3.2 Extending Feature Models with Attributes

The idea of extending (or augmenting) feature models with quality attributes was proposed by many, among them Czarnecki et al. [**28**], Zhang et al. [**115**] and Cordy et al.

[**27**]. Many researchers used feature models with attached attributes and synthetic data to experiment with optimizing feature selection in Software Product Lines, including Benavides et al. [**9**], White et al. [**103**], [**104**], Bagheri et al. [**6**] Soltani et al. [**91**] and Guo et al. [**45**] as will be detailed in section 4.2.

3.3 Our Approach to Assigning Feature Attributes

In order to accommodate user business concerns in the process of product customization, we augmented the feature model with 3 attributes per feature: COST, USED_BEFORE, and DEFECTS. 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 (4)

Table 1 shows a sample of the attributes given to the "Web Portal" feature model.

Table 1: Sample attributes added to feature models

Feature	USED_BEFORE	DEFECTS	COST
add_services	*FALSE*	0	5.0
site_stats	*TRUE*	7	15.0
Logging	*TRUE*	6	9.8
db	*TRUE*	5	11.4

This provides three of the five optimization objectives: to minimize total cost, to maximize feature that were used before, and to minimize the total number of known

defects. The two remaining objectives are: to maximize correctness, i.e. minimize rule violations, and to maximize the number of offered features.

3.4 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 the tools we used (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 features that weren't used before.
4. Known defects; which we seek to minimize.
5. Cost; which we seek to minimize.

3.5 Why Maximize the Number of Features?

Some have looked at the second objective above 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 minimize violations,

new features, known defects, and cost. Such formulation would defeat the purpose of offering a diverse set of valid products.

3.6 Quality of Pareto Front

Quality indicators can be calculated to assess the performance of multi-objective metaheuristics. In this study, we used three indicators:

1. Hypervolume (HV): defined in [**118**] as a measure of the size of the space covered, as follows: Let $(x_1, x_2, \ldots, x_k)$ be a set of decision vectors. The hypervolume is the volume enclosed by the union of the polytopes $(p_1, p_2, \ldots, p_k)$ where each p_i is formed by the intersections of the following hyperplanes arising out of x_i, along with the axes: for each axis in the objective space, there exists a hyperplane perpendicular to the axis and passing through the point $(f_1(x_i), f_2(x_i), \ldots, f_n(x_i))$. In the two-dimensional (2-D) case, each p_i represents a rectangle defined by the points (0,0) and $(f_1(x_i), f_2(x_i))$. In jMetal, all objectives are minimized, but the Pareto front is inverted before calculating hypervolume, thus the preferred Pareto front would be that with the most hypervolume.

2. Spread: defined in [**33**], measures the extent of spread in the obtained solutions. It is found with the formula:

$$\text{Spread} = \frac{d_f + d_l + \sum_{i=1}^{N-1} | d_i - \bar{d} |}{d_f + d_l + (N-1)\bar{d}} \qquad (5)$$

where:

N is the number of points in the Pareto front,

d_i is Euclidean distance among consecutive points,

$\bar{d}$ is Average of d_i's, and

d_f and d_l are the Euclidean distances between the extreme and boundary solutions of Pareto front.

3. %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. For instance, if %Correct = 70% then we have 70 fully-compliant configurations out of 100 members of the final population.

3.7 Run Time versus Number of Evaluations; Which One Shall Be Fixed?

In [**86**], we compared MOEAs by allowing each to perform a fixed number of fitness function evaluations, which is the 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 50K evaluations on 5 objectives for the E-Shop feature model,

which meant that IBEA spent far more time in fitness ranking than NSGA-II. This was expected from our study of the fitness ranking criterion in section 2.7.

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, Spread, %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, regardless of number of evaluations each algorithm were able to perform.

3.8 Relationship with Interactive Configuration

This problem of feature selection in software product lines has the most practical application in user-interactive tools for product configuration. Such tools usually begin with the "bare-bone" product, which has the essential or mandatory features, then allows the user to fix some of the optional features, and makes recommendations regarding the choices to make next. The user may also be prompted to limit the objective space to the levels that they can afford, such as a maximum cost. The tool would periodically take user input and run optimization cycles and come back with recommendations. Multi-objective approaches are best suited for this paradigm since they provide the user with a set of Pareto-optimal choices each with its own trade-offs.

The problem that we attempt to solve in this paper is the most complex form of user-interactive configuration, since we deal with optimum feature selections before the user makes any feature choices or limits the objective space. As the user makes periodic choices, the space of options becomes smaller and smaller, and the tools would perform faster than in the most general case.

Chapter 4: Related Work

4.1 Automated Analysis of Feature Models

In [**75**], 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. The "product enumeration" operation was not attempted for feature models with more than 3 million valid products, since this operation was not feasible. This left out the largest 12 models out of 90, and thus the remaining models were all below 90 features in size.

In [**64**], 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; for instance, the operation took about 27 minutes for a size of 94 features.

In [**70**], 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 [**69**], 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.

The only two studies we know that experimented with the LVAT (Linux Variability Analysis Tools) feature models were done by Johansen et al. [**55**] who generated test covering arrays for feature models, and Henard et al. [**54**] 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. All of the experiments in [**55**] timed out for the largest model (Linux Kernel with 6888 features), and some timed out for even smaller models. In [**54**], the 6-wise coverage experiment took 20 hours for the Linux Kernel feature model.

4.2 Optimizing Feature Models with Attributes

The idea of extending (or augmenting) feature models with quality attributes was proposed by many, among them Czarnecki et al. [**28**], Zhang et al. [**115**] and Cordy et al. [**27**]. The following papers used feature models with attached attributes and synthetic data to experiment with optimizing feature selection in Software Product Lines.

Benavides et al. [**9**] assigned feature attributes such as price range and time range. 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. Clearly, they converted a multi-objective optimization problem into single-objective by combining all the objectives in one formula. They experimented with small models of sizes up to 25 features, and experienced exponentially increasing runtimes.

White et al. [**103**] used feature attributes related to resource consumption, cost and accuracy. They mapped the feature selection problem to a multidimensional multi-choice knapsack problem, and apply Filtered Cartesian Flattening to provide partially optimal feature selection. Then in [**104**], they 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. A sequence of minimal feature adaptations is calculated to reach from the initial to the desired feature model configurations.

Bagheri et al. [**6**] proposed a Stratified Analytic Hierarchy Process, which first

helps to rank and select the most relevant high level business objectives for the target stakeholders (e.g., security over implementation costs), and then helps to rank and select the most relevant features from the feature model to be used as the starting point in the staged configuration process.

Soltani et al. [**91**] annotated features with business concerns and attached a cost to each feature, then employed Hierarchical Task Network (HTN) planning to automatically select suitable features that satisfy the stakeholders' business concerns and resource limitations.

Shi et al. [**88**] utilize a knapsack approximation algorithm along with greedy search and a constraint solver to select features based on customer requirements. They experiment with randomly-generated feature models with various sizes.

More recently, Guo et al. [**45**] augmented feature models with attributes such as memory and CPU usage, and used a Genetic Algorithm to provide an optimum feature selection, thus aggregating all objectives into one formula with preset weights.

In all the works mentioned above (sections 4.1 and 4.2), 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.

4.3 Pareto-Optimal Search-Based Software Engineering

This section is extracted from a recent survey by Sayyad and Ammar [**82**].

Historically, the field of Search-Based Software Engineering (SBSE) has seen a slow adoption of Pareto optimization techniques, generally known as multiobjective optimization techniques. Back in 2001, when Harman and Jones coined the term SBSE [**48**], all surveyed and suggested techniques were based on single-valued fitness functions. In 2007, Harman commented on the current state and future of SBSE [**47**], and in the "Road-map for Future Work" section he suggested using multiobjective optimization. Then in 2009, Harman et al. [**50**] were able to cite several works in which multiobjective optimization techniques were deployed. Still, most of the work reviewed therein, as well as work done thereafter, optimized two objectives only, while higher numbers appeared only occasionally. Also, the typical tendency was to use algorithms that are popular in other domains, such as NSGA-II and SPEA2. In this survey, we attempt to systematically identify these trends.

Several surveys and review papers were published in the field of SBSE [**47**], [**50**], [**51**], [**30**]. Some surveys focused on subfields, such as search-based software design [**77**], and search-based test data generation [**67**]. This survey is the first of its kind, in which we survey SBSE research work that employed Pareto optimization techniques.

The total number of surveyed papers was 51. Figure 9 breaks them down by area of application, while Figure 10 classifies them by year of publication.

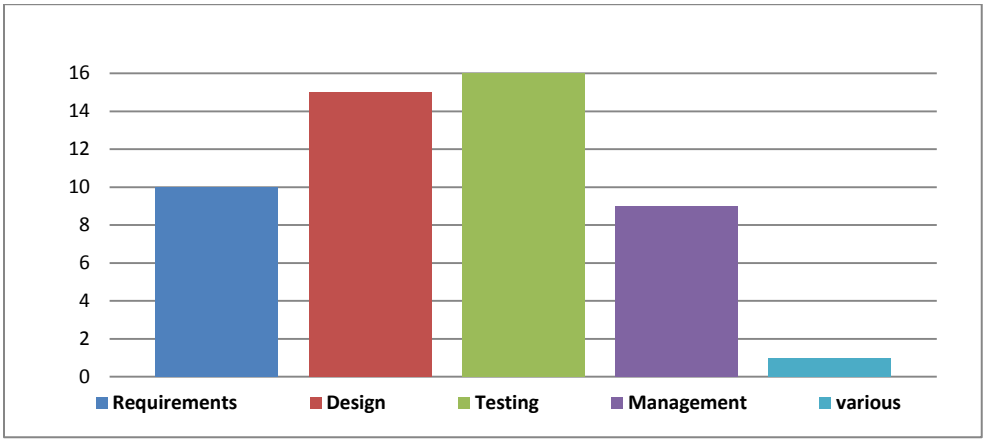

Figure 9: Pareto-Optimal SBSE papers by area of application

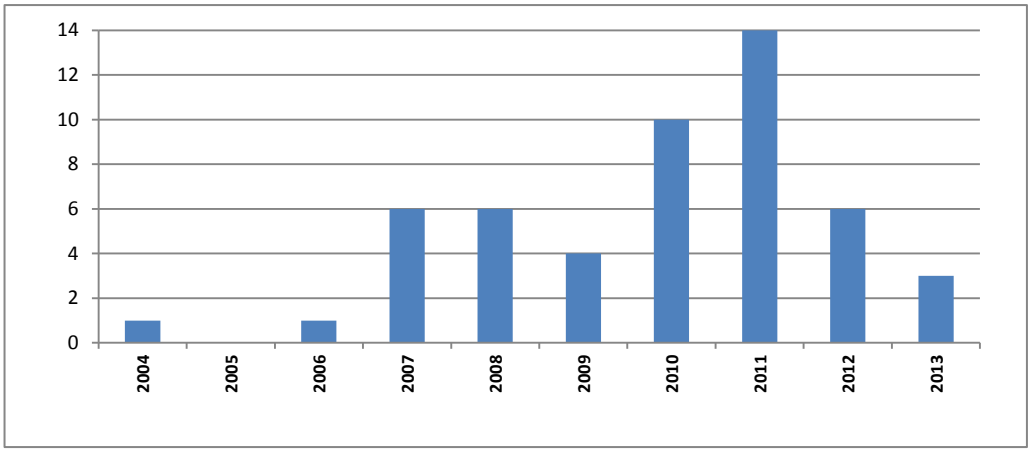

Figure 10: Pareto-Optmal SBSE papers by publication year

Tables 2-5 list all the surveyed papers, along with the multiobjective algorithms, the number of objectives, tools and quality indicators. The surveyed papers are divided according to the application areas of software requirements, design, testing, and management.

Table 2: List of surveyed Pareto-Optimal SBSE works in software requirements

Ref	*Author(s)*	*Year*	*Number of Objectives*	*Algorithm (s)*	*Tool*	*Quality Indicator (s)*
[**114**]	Y. Zhang, M. Harman, S. A. Mansouri	2007	**2**	NSGA-II, Pareto GA	--	--
[**41**]	A. Finkelstein, M. Harman, S Mansouri, J Ren, Y Zhang	2008	**2**	NSGA-II	--	--
[**36**]	J. J. Durillo, Y. Zhang, E. Alba, A. J. Nebro	2009	**2**	NSGA-II, MOCell	jMetal	HV, Spread
[**113**]	Y. Zhang, E. Alba, J. J. Durillo, S. Eldh, M. Harman	2010	**3**	NSGA-II	--	--
[**35**]	J. J. Durillo, Y. Zhang, E. Alba, M. Harman, A. J. Nebro	2011	**2**	NSGA-II, MOCell, PAES	jMetal	HV, Spread
[**18**]	M. M. A. Brasil, T. G. N. da Silva, F. G. de Freitas, J. T. de Souza, M. I. Cortés	2011	**3**	NSGA-II, MOCell	jMetal	HV, Spread
[**53**]	W. Heaven, E. Letier	2011	**2**	NSGA-II	Matlab	--
[**95**]	V. Veerappa, E. Letier	2011	**2**	GA with Clustering	Matlab	--
[**60**]	A.C. Kumari, K. Srinivas, M.P. Gupta	2012	**2**	QEMEA	--	HV, Spread, Convergence
[**86**]	A. S. Sayyad, T. Menzies, H. Ammar	2013	**2, 3, 4, 5**	IBEA, NSGA-II, ssNSGA-II, SPEA2, FastPGA, MOCell, MOCHC	jMetal	HV, Spread

Table 3: List of surveyed Pareto-Optimal SBSE works in software design

Ref	*Author(s)*	*Year*	*Number of Objectives*	*Algorithm (s)*	*Tool*	*Quality Indicator (s)*
[**58**]	T. Khoshgoftaar, Y. Liu, N. Seliya	2004	**4**	NSGA-II	--	--
[**43**]	L. Grunske	2006	**2**	MOGA	--	--
[**63**]	Z. Liu, H. Guo, D. Li, T. Han, J. Zhang	2007	**5**	MOGA	Frontier	--
[**52**]	M. Harman, L. Tratt	2007	**2**	Pareto Hill Climbing	--	--
[**4**]	A. Arcuri, D. White, J. Clark, X. Yao	2008	**2**	SPEA2	--	--
[**100**]	J. Wang, Y. Hou	2008	**2**	MOGA	--	--
[**97**]	H. Wada, P. Champrasert, J. Suzuki, K. Oba	2008	**10**	E^3-MOGA	--	--
[**89**]	C. L. Simons, I. C. Parmee	2008	**2**	NSGA-II	--	--
[**90**]	C. L. Simons, I. C. Parmee, R. Gwynllyw	2010	**3**	NSGA-II	--	--
[**17**]	M. Bowman, L. C. Briand, Y. Labiche	2010	**5**	SPEA2	--	--
[**78**]	O. Räihä, K. Koskimies, E. Mäkinen	2011	**2**	MOGA	--	--
[**76**]	K. Praditwong, M. Harman, X. Yao	2011	**5**	Two-Archive MOEA	--	--
[**7**]	M. O. Barros	2012	**4, 5**	NSGA-II	jMetal	GD, Error Ratio
[**24**]	T. E. Colanzi, S.R. Vergilio	2012	**5**	NSGA-II	--	--
[**107**]	J. L. Wilkerson, D. R. Tauritz, J. M. Bridges	2102	**4**	NSGA-II	--	--
[**42**]	S. Frey, F. Fittkau, W. Hasselbring	2013	**3**	NSGA-II	Opt4J	HV, IGD

Table 4: List of surveyed Pareto-Optimal SBSE works in software testing

Ref	*Author(s)*	*Year*	*Number of Objectives*	*Algorithm (s)*	*Tool*	*Quality Indicator (s)*
[**109**]	S. Yoo, M. Harman	2007	**2, 3**	NSGA-II, vNSGA-II	--	--
[**49**]	M. Harman, K. Lakhotia, P. McMinn	2007	**2**	NSGA-II	--	--
[**101**]	Z. Wang, K. Tang, X. Yao	2008	**2, 3**	NSGA-II, MODE	--	--
[**66**]	C. Maia, R. Carmo, F. Freitas, G. Campos, J. Souza	2009	**3**	NSGA-II	jMetal	--
[**108**]	T. Yano, E. Martins, F. Sousa	2010	**2**	M-GEO$_{vsl}$	--	--
[**73**]	G. H. L. Pinto, S. R. Vergilio	2010	**3**	NSGA-II	--	--
[**61**]	W. Langdon, M. Harman, Y. Jia	2010	**2**	NSGA-II	--	--
[**110**]	S. Yoo, M. Harman	2010	**2, 3**	HNSGA-II	--	--
[**102**]	Z. Wang, K. Tang, X. Yao	2010	**2, 3**	NSGA-II, HaD-MOEA	--	HV
[**20**]	R. Cabral, A. T. R. Pozo, S. R. Vergilio	2010	**2**	Pareto Ant Colony	--	--
[**23**]	T. E. Colanzi, W. Assuncao, S. R. Vergilio, A. Pozo	2011	**2**	NSGA-II, SPEA2	jMetal	GD, ED
[**5**]	W. K. G. Assunção, T. E. Colanzi, A. T. R. Pozo, S. Vergilio	2011	**4**	NSGA-II, SPEA2	jMetal	GD, IGD, Coverage, ED
[**112**]	S. Yoo, R. Nilsson, M. Harman	2011	**3**	Two-Archive MOEA	--	--
[**111**]	S. Yoo, M. Harman, S. Ur	2011	**2**	NSGA-II	jMetal	--
[**39**]	J. Ferrer, F. Chicano, E. Alba	2011	**2**	NSGA-II, SPEA2, PAES, MOCell, Random	jMetal	HV, attainment surfaces

Table 5: List of surveyed Pareto-Optimal SBSE works in project management

Ref	*Author(s)*	*Year*	*Number of Objectives*	*Algorithm (s)*	*Tool*	*Quality Indicator (s)*
57]	T. Khoshgoftaar, Y. Liu	2007	**3**	NSGA-II	--	--
[**44**]	S. Gueorguiev, M. Harman, G. Antoniol	2009	**2**	SPEA2	--	--
[**99**]	Z. Wang, T. Chen, K. Tang, X. Yao	2009	**2**	NSGA-II	--	--
[**59**]	T. Kremmel, J. Kubalik, S. Biffl	2011	**5**	mPOEMS, NSGA-II, SPEA2	--	HV, Coverage
[**32**]	J. T. de Souza, C. L. Maia, F. G. de Freitas, D. P. Coutinho	2010	**2**	NSGA-II, MOCell, Random	jMetal	HV, Spread, Coverage
[**22**]	F. Chicano, F. Luna, A. J. Nebro, E. Alba	2011	**2**	NSGA-II, SPEA2, PAES, MOCell, GDE3	jMetal	HV, attainment surfaces
[**79**]	D. Rodríguez, M. Ruiz, J. C. Riquelme, R. Harrison	2011	**2, 3, 5**	NSGA-II	jMetal	--
[**19**]	R. Britto, P. S. Neto, R. Rabelo, W. Ayala, T. Soares	2012	**2**	NSGA-II	--	--
[**81**]	F. Sarro, F. Ferrucci, C. Gravino	2012	**5**	MOGP	--	--
[**40**]	F. Ferrucci, M. Harman, J. Ren, F. Sarro	2013	**3**	NSGA-II, NSGA-II$_v$	--	HV, GD, Coverage

We now analyze and provide commentary about the data collected from the surveyed papers with regard to algorithms, number of objectives, tools, and quality indicators.

4.3.1 Algorithms

The total number of different algorithms used in all 51 papers was 25 algorithms. Figure 11 shows the frequency of use for each of the most used algorithms. The "other" category includes 20 algorithms that were used only once or twice in all the surveyed papers.

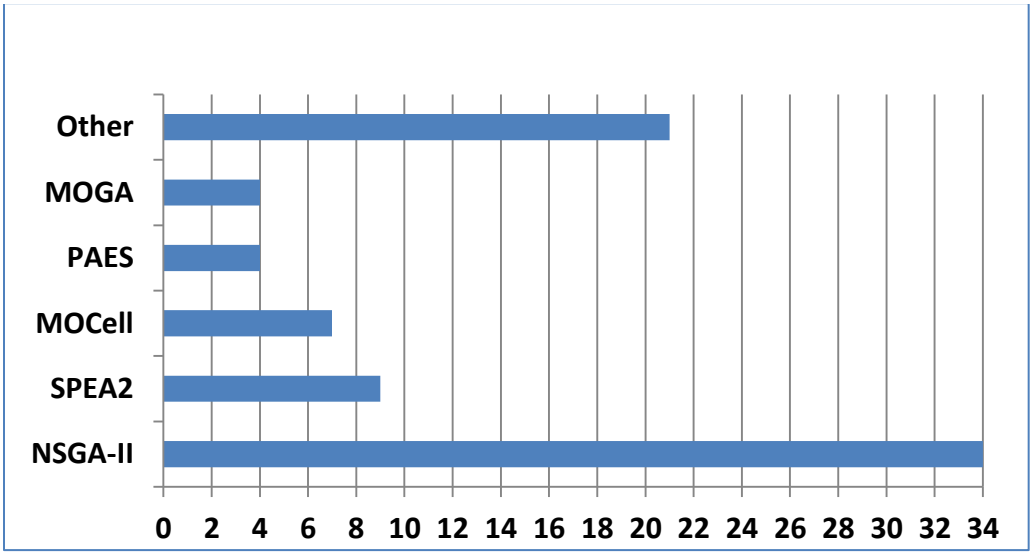

Figure 11: Pareto-Optimal SBSE algorithms by frequency of use

We observe that 36 papers (70%) used a single algorithm, whereas 15 papers (30%) used multiple algorithms for comparison purposes.

Figure 12 shows the frequency of use in the papers that only used a single Pareto optimization algorithm. NSGA-II was the algorithm of choice in 53% of those papers.

The “other” category includes 9 algorithms that were used only once or twice in this category of papers.

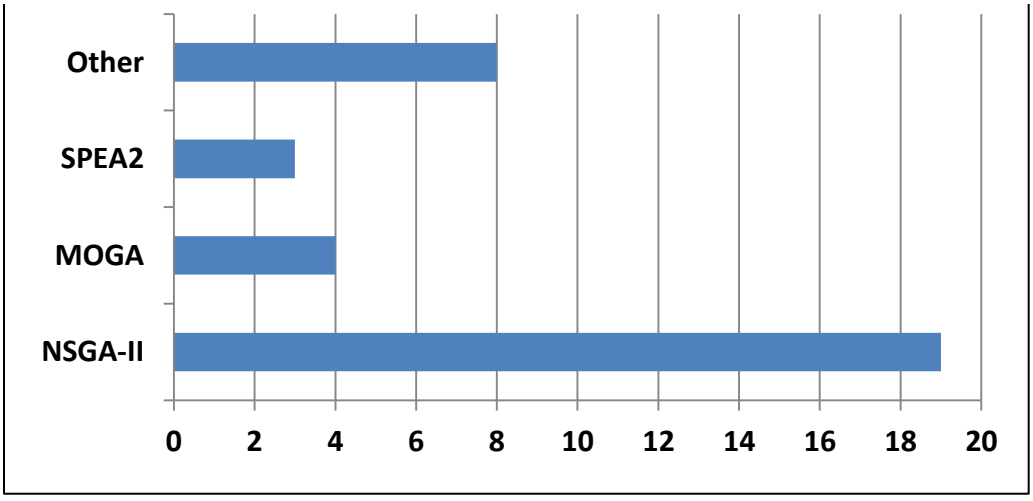

Figure 12: Pareto-Optimal SBSE algorithms by frequency of use (single-algo. papers)

We notice that MOCell and PAES appear nowhere in this category. They were only used in comparison to other algorithms, but never alone as the algorithm of choice.

For those 35 papers that only used one algorithm, we pose an important question: why choose that particular algorithm? The answers are charted in Figure 13.

Most researchers didn’t state any reason for adopting a certain MOEA (42%) or stated popularity of the algorithm (particularly NSGA-II) as the sole reason (25%). This is usually justified by the fact that many of those papers introduced Pareto-optimal solutions to their problems for the first time, which in itself is considered a contribution. A sizeable portion of these papers (10, 28%) introduced new MOEAs to solve a problem, but only compared their outcome with that of single-objective algorithms. Looking at the

15 papers that actually compared MOEAs to one another, 5 of them introduced new MOEAs and compared their outcomes with popular MOEAs.

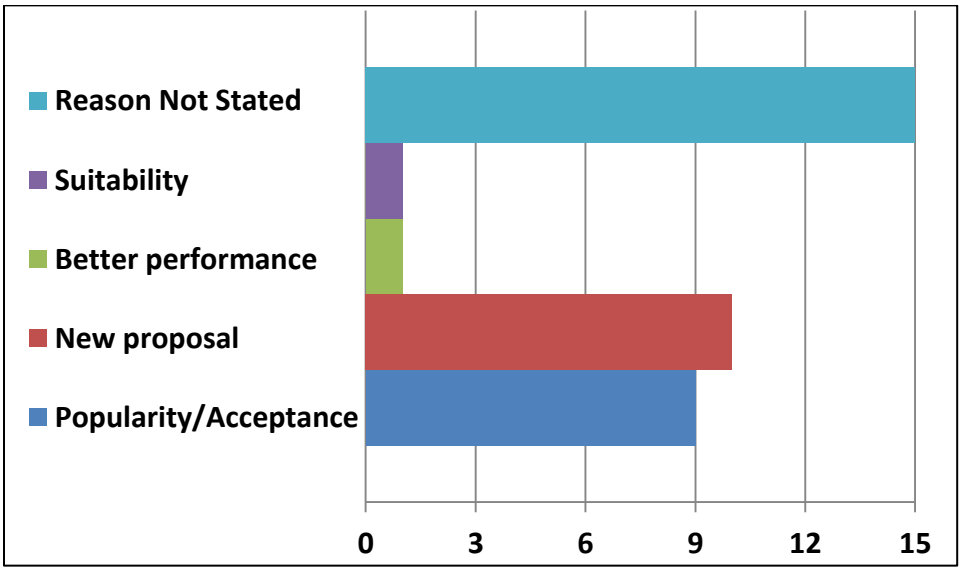

Figure 13: Reasons for adopting an algorithm in Pareto-Optimal SBSE

In the single-MOEA papers, only one stated that the MOEA was chosen because it was more suitable for the particular problem, and one paper stated that the chosen algorithm had been reported to have better performance in higher-dimension objective spaces.

4.3.2 Number of Objectives

The first step in Pareto optimization is to re-formulate the problem to bring out the competing objectives. Although the objectives were known all along, legacy research combined them into one fitness function. Pareto optimization researchers then had to decide which objectives to represent as separate, which to combine into one, and which to

ignore; thus a different number of objectives may have been defined for the same problem. Figure 14 shows the variety of "number of objectives in the surveyed papers. There were 7 papers that presented different formulations for each problem, with a different number of objectives in each formulation.

4.3.3 Tools/Frameworks

17 papers (33%) reported using implementations of the algorithms that are available in tools such as jMetal (13 papers) and Matlab (2 papers). Two thirds of the time, the researchers had to code their own implementations, which is the assumption we made when tools weren't mentioned.

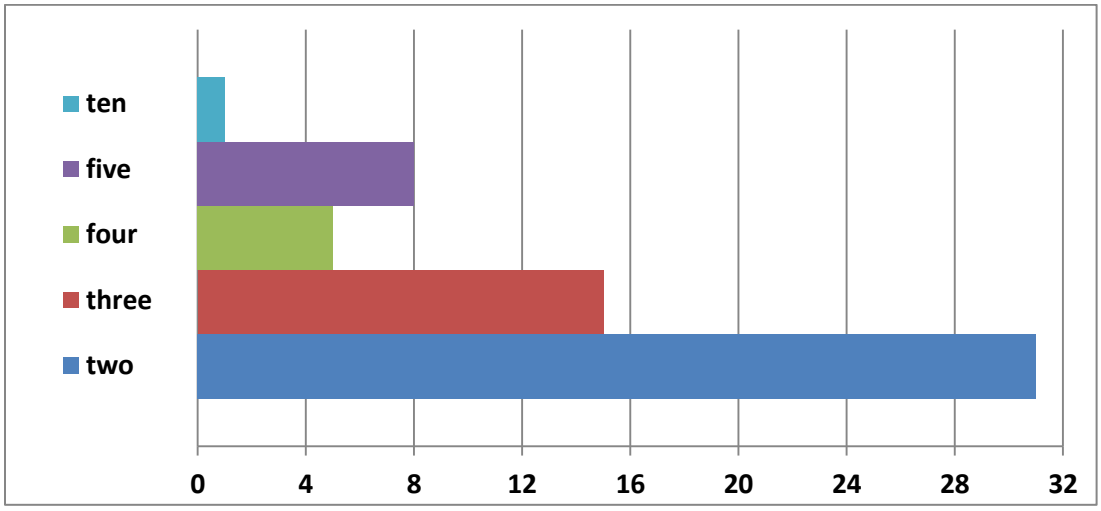

Figure 14: Number of objectives by frequency of use in Pareto-Optimal SBSE

4.3.4 Quality Indicators

15 papers (30%) used quality indicators to assess the quality of the Pareto fronts and, most of the time (12 papers), to compare the performance of various MOEAs against

one another. Hypervolume (HV) was the most widely used indicator (12 papers), while there was lesser agreement on the use of other indicators.

Quality indicators are useful as aggregate measures for large sets of solutions, especially with higher dimensions in the objective space. Skipping the computation of quality indicators is understandable when it's possible to directly assess the objective values, which is the case when the Pareto front can be charted in 2-D or 3-D.

4.3.5 Survey Summary

We surveyed 51 research papers that applied multiobjective search-based optimization methods to software engineering problems. This being a relatively young trend in SBSE, we have observed certain shortcomings in many papers:

1. Lack of clarity regarding the reasons why an algorithm is chosen for a problem (Figure 5).

2. Tendency to simplify problems by specifying fewer objectives to evaluate (Figure 6),

3. Heavy reliance on personal implementations of widely-used algorithms (Table 1, the "Tool" column).

4. Lack of agreement on whether to utilize quality indicators, and which indicators to use (Table 1, the "Quality Indicators" column).

But we also noticed some promising directions:

1. Researchers are comparing algorithms against one another to discover better performance, and to reason about suitability of the algorithms to the problems at hand (15 papers out of 51).

2. Some papers are exploring different formulations of their problems wherein the complexity is increased and more objectives are evaluated (7 papers out of 51).

3. Increasing use of the open-source jMetal framework with its rich set of MOEAs and quality indicators (13 papers out of 51).

Chapter 5: Experimental Setup

5.1 Feature Models Used in this Study

The feature models used in this study belong to two feature model repositories: SPLOT [**68**] and LVAT [**12**]. From SPLOT, we chose 20 non-trivial feature models with varying sizes (between 43 and 290 features), and we made sure that we chose models with cross-tree constraints (CTCs). The majority of feature models in SPLOT are below 43 features in size, and many do not have any CTCs.

Table 6 shows the feature models, along with their sizes (number of features), CTCs, and total number of rules. For an explanation of "features", "CTCs", and "total rules", please refer to section 2.2.

Table 6: SPLOT feature models used in the study

ID	Feature model	Features	CTCs	Total rules
FM-43	Web portal	43	6	63
FM-43B	Mobile Media 2	43	3	65
FM-44	Documentation Generation	44	8	68
FM-45	Android SPL	45	5	74
FM-46	DELL Notebook Computers	46	110	171
FM-52	Linea de Experimentos	52	4	87
FM-53	Video Player	53	2	78
FM-60	Smart Home v2.2	60	2	82
FM-61	Arcade Game PL	61	34	122
FM-63	OW2-FraSCAti-1.4	63	46	129
FM-66	bCMS system	66	2	109
FM-67	HIS	67	4	121
FM-70	DATABASE TOOLS	70	2	82
FM-72	Car Selection	72	18	123
FM-72B	Reuso - UFRJ - Eclipse1	72	1	97
FM-88	Billing	88	59	160
FM-94	Coche ecologico	94	2	151
FM-97	UP structural	97	1	138
FM-137	xtext	137	1	179
FM-290	E-Shop	290	21	426

From LVAT, we apply our methods to 7 models with sizes between 544 and 6888 features, as shown in Table 7.

Table 7: LVAT feature models used in the study

Model	Version	Features	Rules
ToyBox	0.1.0	544	1020
axTLS	1.2.7	684	2155
eCos	3.0	1244	3146
FreeBSD	8.0.0	1396	62183
Fiasco	2011081207	1638	5228
uClinux	20100825	1850	2468
Linux Kernel	2.6.28	6888	343944

5.2 Problem Encoding

For the purpose of running the MOEAs to configure the feature models, 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.

5.3 Algorithm Implementation

We utilized the implementations of metaheuristic algorithms within the jMetal framework [**34**], an open-source package in Java with a rich set of multi-objective evolutionary algorithms, benchmark problems, quality indicators, and experimentation routines.

5.4 Strong Heuristics to Improve Performance

Next we describe the three strong heuristic techniques that we utilized to help the MOEAs arrive faster at correct configurations and thus save on computation time and improve performance. Those strong heuristics are: PUSH, PULL, and population seeding.

5.4.1 The PUSH Technique

In the PUSH technique, we use knowledge of the feature models structure to restrict the evolutions process. Since the SPLOT models are represented differently from the LVAT models, the PUSH technique is manifested differently. For SPLOT, we "Tree Mutation," whereas for LVAT we use "feature fixing."

Figure 15 lists the procedure for tree mutation, used with SPLOT feature models. This new operator honors and preserves the tree structure in the feature model.

```
For each bit in the decision string
  if rand(0,1) < mutation_probability:
    Don't mutate if:
      1) deselecting root feature,
      2) selecting feature whose parent is not selected,
      3) deselecting a mandatory child feature whose
      parent is selected, or
      4) group cardinality is violated
    Otherwise:
      flip this bit, and
      if selecting (turning on) a feature then:
            turn on children (a minimum skeleton)
      else if deselecting (turning off) a feature then:
        turn off all children
```

Figure 15: Tree mutation procedure

Violations of cross-tree constraints (CTCs) still need to be minimized by the MOEA as an optimization objective. Thus the fitness evaluation function only checks violations of the CTCs, as opposed to our original approach [**86**] of counting violations of all the rules in the feature model, including the CTCs.

Feature fixing was introduced in [**85**]. In the DIMACS formulas representing LVAT 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 is 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 8 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.

5.4.2 The PULL Technique

In the PULL technique, we give constraint violations 4 times the weight as each one of the other 4 objectives, thus "pulling" the optimization process towards satisfying the feature model constraints first, then proceeding to find Pareto-optimal configurations according to the user preferences from within the space of valid solutions.

Table 8: Fixed features and skipped rules for LVAT models

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 Kernel	6888	94	343944	699

5.4.3 Population Seeding

This additional strong heuristic builds on the ability of "fit" individuals to influence other individuals in the population towards learning the desired qualities. In this approach, one correct configuration is pre-computed and then planted in the initial population for the 5-objective optimization. The desired effect is to guide an MOEA into finding a range of solutions that can be suggested to the user in the initial stage of interactive configuration. Different techniques were employed to generate correct seeds, with varying results, as will be shown in the next chapter.

Chapter 6: Results

6.1 Increasing the Number of Optimization Objectives

Our first experiment was reported in [**86**], where we examined the effect of varying the number of optimization objectives on the performance of Multi-Objective Evolutionary Algorithms (MOEAs).

Table 9 lists the configuration parameters that were used in this experiment. The settings used here were mostly the default setting in jMetal. The focus was on comparing MOEAs to one another, rather than tuning the parameters to achieve the best performance, which is explored in the next experiment.

Table 9: Parameter settings for first experiment

Parameter	Setting
Population size	100
Crossover type	Single-Point Crossover
Crossover probability	0.9
Mutation type	Bit-Flip Mutation
Mutation probability	0.05

As listed in section 3.4, the optimization objectives are:

1. Correctness; i.e. compliance to the relationships and constraints defined in the feature model. Since the tools we used (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 features that weren't used

before.

4. Known defects; which we seek to minimize.
5. Cost; which we seek to minimize.

The variation of "number of objectives" in this experiment was as follows:

1. In the two-objective run, we optimize objectives 1 and 2 above, i.e. correctness and number of features.
2. In the three-objective run, we optimize objectives 1, 2 and 5 above.
3. In the four-objective run, we optimize objectives 1, 2, 3 and 5 above.
4. In the five-objective run, we optimize all the objectives mentioned above.

The results of the experiment are shown in Tables 10 and 11. In the 50-K runs, each algorithm run is repeated 10 times, each time making 50,000 evaluations of the objective functions. The median of the quality indicators (HV, Spread and %Correct) is reported. In the 50-M runs, each algorithms is run only once with 50 Million evaluations. Black cells with white font indicate results where IBEA performed much better than other algorithms, as discussed below.

Looking at the results in Tables 10 and 11, we make the following observations:

1- IBEA stands out in terms of HV, spread, and percentage of fully-correct solutions, as can be seen in the black cells with white font. The remarkable gains with this algorithm become obvious with the bigger, more complex E-Shop feature model, and with more objectives.

Table 10: Comparison of quality indicators, 2 and 3 objectives

FM	ALGORITHM	3 Objectives			2 Objectives		
		HV	SPRD	% CR	HV	SPRD	% CR
Web Portal (50K evals)	IBEA	0.56	1.33	**82%**	1.00	1.04	14%
	NSGA-II	0.57	1.03	24%	1.00	1.04	14%
	MOCell	0.57	1.07	27%	1.00	0.95	20%
	FastPGA	0.57	1.13	34%	1.00	1.04	14%
	SPEA2	0.57	0.86	13%	1.00	1.04	14%
E-Shop (50M evals)	IBEA	0.52	1.08	**80%**	1.00	1.00	100%
	NSGA-II	0.42	0.93	1%	1.00	1.00	100%
	MOCell	0.43	0.76	1%	1.00	1.00	100%
	FastPGA	0.42	0.85	1%	1.00	1.00	100%
	SPEA2	0.39	0.60	0%	1.00	1.00	100%

Table 11: Comparison of quality indicators, 4 and 5 objectives

FM	ALGORITHM	5 Objectives			4 Objectives		
		HV	SPRD	% CR	HV	SPRD	% CR
Web Portal (50K evals)	IBEA	0.34	0.79	**73%**	0.43	1.18	**56%**
	NSGA-II	0.27	0.68	8%	0.39	0.74	8%
	MOCell	0.27	0.65	4%	0.38	0.68	8%
	FastPGA	0.27	0.68	7%	0.40	0.71	9%
	SPEA2	0.27	0.54	0%	0.40	0.62	2%
E-Shop (50M evals)	IBEA	0.28	0.88	**52%**	0.37	1.15	**42%**
	NSGA-II	0.23	0.75	1%	0.25	0.81	1%
	MOCell	0.23	0.82	1%	0.25	0.86	1%
	FastPGA	0.22	0.82	0%	0.25	0.81	1%
	SPEA2	0.20	0.55	0%	0.23	0.64	0%

2- We pay special attention to the better spread results that IBEA achieves, although IBEA does not incorporate crowd pruning measures in its dominance criteria. It outperforms the very algorithms that depend on crowd pruning.

3- The remaining algorithms (NSGA-II, SPEA2, FastPGA, and MOCell) perform moderately with the smaller Web Portal feature model with 2 and 3 objectives. With 4 or 5 objectives, or when applied to the bigger E-Shop feature model, these algorithms do not perform nearly acceptably when compared with the IBEA results. The least performing among these was SPEA2, especially with 4 and 5 objectives.

4- In the 2-objective run for E-Shop, with 50 Million evaluations, all algorithms seem to yield the same results, with 100% of the Pareto front having zero violations. After examining the resulting Pareto fronts, we find that IBEA outperforms in this category as well; since IBEA is able to find 3 solutions, each having zero violations and only one missing feature. Other algorithms only found one or two solutions, and some solutions had two missing features.

In order to get a rough idea about how soon IBEA arrives at fully-correct solutions, we performed further runs on the E-Shop feature model with 5 objectives. The results are shown in Table 12.

Table 12: Various runs with IBEA on E-Shop, 5 objectives

Evals.	Run Time	HV	Spread	% Correct
50,000	12.3 sec	0.21	0.77	0%
1 Million	4 min	0.25	0.77	0%
2 Million	8 min	0.25	0.98	3%
5 Million	20 min	0.27	1.00	16%
10 Million	38 min	0.28	0.86	16%
20 Million	81 min	0.28	0.91	30%
50 Million	~ 3 hours	0.28	0.88	52%

The quickest results were 3 compliant configurations obtained within 8 minutes. Such results can be useful when a reasoned argument is needed quickly (e.g. at a meeting). Longer runs would be needed to get more optimized and better diversified results. While IBEA achieved 3 acceptable configuration in about 8 minutes, other algorithms took 50 million evaluations (about 3 hours) to find just one acceptable configuration.

6.2 Parameter Tuning

Next we present a parameter tuning experiment which was reported in [**83**], where we examined the effect of different combinations of parameters on the performance of IBEA and NSGA-II.

As mentioned in the previous section, the parameter settings used were the default settings in jMetal, which conform with the usual default setting recommended in the literature, such as in [**37**]. Researchers tend to take the default settings for granted, but recent studies have shown the importance of parameter tuning and the influence of parameter choice on performance, such as the study by Arcuri and Fraser [**3**].

In this experiment, we consider 3 research questions. RQ1 and RQ2 were considered by Arcuri and Fraser [**2**]. We added RQ3 since we're interested in comparing IBEA to NSGA-II.

RQ1: How Large is the Potential Impact of a Wrong Choice of Parameter Settings?

RQ2: How Does a "Default" Setting Compare to the Best and Worst Achievable Performance?

RQ3: How does the performance of IBEA's best tuning compare to NSGA-II's best tuning?

Next, we explain the experimental setup and the value choices for the different parameters. Where possible, we used the same parameter levels that were used by Arcuri and Fraser. In some cases, we used fixed values because the value levels were not available in jMetal. Table 13 shows the similarities and differences between Arcuri and Fraser and this study.

Table 13: Experimental setup for parameter tuning

Parameter	***Arcuri and Fraser [2]***	***This experiment***
Algorithm	Genetic Algorithm	NSGA-II, IBEA
Mutation rate	0.1	{0, 0.5, 1, 1.5, 2}/Features
Crossover rate	{0 , .2 , .5 , .8 , 1}	{0 , .2 , .5 , .8 , 1}
Population size	{4 , 10, 50 , 100 , 200}	{10, 50 , 100 , 150, 200}
Elitism rate	{0 , 1, 10% , 50%} or steady state	Elitism rate does not apply to either NSGA-II or IBEA. Steady state is implemented in jMetal for NSGA-II but not IBEA. Thus no variation will be considered.
Selection	roulette wheel, tournament with size either 2 or 7, and rank selection with bias either 1.2 or 1.7	Binary tournament only.
Repeats	15	15
Test bed	20 classes	20 feature models
Number of configurations	5 x 5 x 5 x 5 x 2 = 1250	2 x 5 x 5 x 5 = 250

Each feature model (FM) was operated on for 10 seconds. The overall time to execute our experiment was = 250 configurations x 15 runs x 20 FMs x 10 sec = 12500 minutes = 208.3 hours = ~8.7 days.

The default parameter values according to jMetal are: Population = 100, Crossover rate = 0.8, and Mutation rate = 1/FEATURES.

In this experiment, we restrict each algorithm to 10 seconds per feature model, rather than restricting the number of objective function evaluations. For justification of this, please refer to section 3.7.

We performed the same analysis done by Arcuri and Fraser [**2**], in which they performed two-way comparisons composed of two parts: effect size and statistical significance.

For effect size, the Vargha-Delaney A measure [**94**] was used, as implemented in R by Thomas et al [**93**]. It tells us how often, on average, one technique outperforms the other. When applied to two populations such as the results of two techniques, the A measure is a value between 0 and 1: when the A measure is exactly 0.5, then the two techniques achieve equal performance; when A is less than 0.5, the first technique is worse; and when A is more than 0.5, the second technique is worse. The closer to 0.5, the smaller the difference between the techniques; the farther from 0.5, the larger the difference. [**93**]

For statistical significance, the Mann-Whitney test was used as implemented in R.

Table 14 shows a summary of the parameter tuning results. For each feature model (FM), we show the average over 15 runs of the percentage of correct configurations (%correct) found by best parameter setting and the default parameter setting for both IBEA and NSGA-II. Then we calculate three effect size values:

1. $\hat{A}_{BDI}$: Vargha-Delaney effect size of best compared to default for IBEA.
2. $\hat{A}_{BDN}$: Vargha-Delaney effect size of best compared to default for NSGA-II.
3. $\hat{A}_{BIN}$: Vargha-Delaney effect size of best IBEA compared to best NSGA-II.

We also calculate the Mann-Whitney statistical significance measure, and we highlight the A-measure in bold when the Mann-Whitney p-value is less than 5%.

The results in table 3 show the following:

4. For both IBEA and NSGA-II, the best parameter tuning beats the default parameter values by a large and significant amount in 15 out of 20 models. For the other 5 models there is improvement, but it is neither large nor statistically significant.

5. The best IBEA results beat the best NSGA-II results in 19 out of 20 models. The improvement was large and significant in 17 out of 19 models. In the one case where NSGA-II results beat IBEA results, the improvement was neither large nor significant.

Thus we are able to answer the first 3 research questions:

RQ1: How Large is the Potential Impact of a Wrong Choice of Parameter Settings?

We confirm Arcuri and Fraser's [**2**] conclusion: *Different parameter settings cause very large variance in the performance*.

Table 14: Summary of parameter tuning results

FM	*Best IBEA*	*Default IBEA*	*Best NSGA-II*	*Default NSGA-II*	$\hat{A}_{BDI}$	$\hat{A}_{BDN}$	$\hat{A}_{BIN}$
FM-43	98%	91%	20%	7%	0.63	**1**	**1**
FM-43B	48%	43%	14%	1.3%	0.58	**1**	**1**
FM-44	60%	51%	13%	1.4%	**0.76**	**1**	**1**
FM-45	27%	11%	9%	0.8%	**0.88**	**1**	**1**
FM-46	5%	0%	0.3%	0%	**0.8**	**0.80**	0.68
FM-52	22%	2%	23%	7%	**0.95**	**1**	0.55
FM-53	79%	71%	27%	16%	0.53	**1**	**1**
FM-60	85%	62%	14%	11%	0.61	**0.86**	**1**
FM-61	39%	9%	21%	5%	**0.90**	**1**	**0.86**
FM-63	54%	19%	11%	1.0%	**0.98**	**1**	**1**
FM-66	14%	0%	5%	0.3%	**0.9**	0.66	**0.85**
FM-67	14%	0%	5%	0.6%	**0.93**	0.59	**0.86**
FM-70	100%	100%	10%	2%	0.5	**1**	**1**
FM-72	27%	13%	0.7%	0.2%	**0.82**	0.47	**1**
FM-72B	97%	66%	10%	1.0%	0.64	**1**	**1**
FM-88	27%	0%	12%	0.3%	**0.90**	**1**	**0.80**
FM-94	14%	0%	0.7%	0%	**0.97**	0.53	**0.95**
FM-97	67%	22%	3%	0.1%	**0.82**	0.61	**0.96**
FM-137	96%	23%	23%	0.3%	**0.86**	**1**	**1**
FM-290	37%	0%	22%	0%	**0.73**	**1**	0.45

RQ2: How Does a "Default" Setting Compare to the Best and Worst Achievable Performance?

Arcuri and Fraser [**2**] concluded that: Default parameter settings perform relatively well, but are far from optimal on individual problem instances.

With our results from table 3, we are able to make a stronger conclusion: *Default parameter settings perform generally poorly, but might perform relatively well on individual problem instances*.

RQ3: How does the performance of IBEA's best tuning compare to NSGA-II's best tuning?

Our results show that *IBEA's best tuning performs generally much better than NSGA-II's best tuning*. This RQ is of no concern to Arcuri and Fraser [**2**], but it confirms our early findings (section 6.1). Specifically, we verify here that no parameter settings enable NSGA-II to achieve acceptable levels of correctness and optimality for the leaned configurations. This result was expected since we have identified the core fitness assignment method as the reason for IBEA's advantage over NSGA-II and other Pareto-based algorithms. IBEA is able to exploit the user preferences in ranking the solutions for selection to generate new solutions and to survive through the evolutionary process; whereas NSGA-II depends on absolute dominance as the deciding criterion, coupled with crowd pruning as a mechanism to force solution diversity, which ignores rich details from the user preferences. This was explained in section 2.7.

In addition to the assertions above, we found that the best parameter tuning when all 20 feature models are considered is: {IBEA, Population = 50, Crossover rate = 0, Mutation rate = 0.5/FEATURES}. This suggests that crossover does more harm than good, since it does not conform with the tree structure of the feature models, i.e. crossover can add a subfeature to a solution in which the parent feature is not selected, and so forth. It also suggests a very low rate for mutation, 0.5/FEATURES, which means

to change one feature only in half the population on average. This supports more exploitation of the model than exploration, and emphasizes the fine grained nature of the feature models, where small changes can have large effects in the optimization process.

6.3 Using Strong Heuristics: PUSH and PULL

So far we have not used any techniques to help IBEA (or other algorithms) to converge faster to correct configurations. In this section, we will apply the PUSH technique (described in subsection 5.4.1) and the PULL technique (described in subsection 5.4.2). We apply IBEA and NSGA-II to all 27 feature models, 20 from SPLOT and 7 from LVAT, as listed in section 5.1.

Table 15 shows the parameter settings used in this experiment:

Table 15: Parameter settings used for experiment with PUSH and PULL

Parameter	Setting
Population size	100
Archive size	100
Crossover type	No Crossover
Mutation type	Bit-Flip Mutation
Mutation probability	1/Number of Features
Runtime	5 minutes
Independent runs	10

First, we show the results of applying PUSH without PULL. Table 16 shows the median values for the quality indicators (%Correct, Hypervolume, and Spread) after applying IBEA and NSGA-II to each of the feature models 10 independent times, each time stopping after 5 minutes. The median value is highlighted in bold if it is statistically

Table 16: Results of 5-objective optimization; PUSH without PULL

FM	IBEA			NSGA-II		
	%CR	HV	SPRD	%CR	HV	SPRD
FM-43	**100%**	0.31	0.81	54%	0.31	**0.88**
FM43B	**100%**	0.23	0.76	59%	0.22	**0.84**
FM-44	**100%**	0.25	0.75	34%	0.25	**0.83**
FM-45	**100%**	0.15	**0.85**	51%	0.15	0.73
FM-46	**98%**	0.25	**1.03**	46%	0.26	0.74
FM-52	100%	0.03	0.63	100%	0.04	**1.04**
FM-53	**100%**	0.22	0.64	61%	0.21	**0.91**
FM-60	100%	0.28	0.60	73%	0.27	**0.95**
FM-61	**63%**	0.12	**0.99**	23%	0.12	0.87
FM-63	**100%**	**0.20**	0.62	30%	0.18	**0.91**
FM-66	100%	0.05	1.03	100%	**0.09**	**1.26**
FM-67	**100%**	0.08	0.78	51%	0.08	0.79
FM-70	100%	**0.35**	0.66	82%	0.30	**0.98**
FM-72	**100%**	0.30	0.71	42%	0.29	**0.95**
FM72B	**100%**	**0.32**	0.58	68%	0.30	**1.01**
FM-88	**20%**	0.24	**0.96**	4%	0.23	0.89
FM-94	**100%**	0.17	0.64	55%	0.17	**0.85**
FM-97	100%	0.27	0.58	94%	0.26	0.95
FM137	**100%**	**0.32**	0.62	75%	0.28	**0.94**
FM290	**100%**	**0.24**	0.65	8%	0.21	**1.01**
ToyBox	**100%**	0.18	0.63	5%	0.20	**0.91**
axTLS	**100%**	0.20	0.62	6%	0.20	**0.90**
eCos	**100%**	**0.30**	0.69	3%	0.10	**0.98**
FreeBSD	**54%**	**0.18**	**1.22**	0%	0.00	1.13
Fiasco	**100%**	0.18	0.67	4%	0.17	**0.95**
uClinux	**33%**	**0.28**	0.59	6%	0.14	**1.03**
Linux Kernel	0%	**0.021**	1.08	0%	0	1.09

better than its counterparts for other algorithms applied to the same feature model, according to the Mann-Whitney test with 95% strength.

The results of Table 16 show the following:

1- Comparing the %Correct columns, we find that IBEA, which employs

continuous dominance criteria, outperforms NSGA-II, which relies on Boolean dominance. This is true for 24 out of 27 models, and becomes more obvious as models increase in size and complexity, except for the largest model (Linux Kernel) where no correct solutions are found within 5 minutes. Overall, IBEA is able to achieve 100% correctness (100 fully-compliant solutions in the final population) in 21 out of 27 models, and the IBEA advantage is statistically significant in 21 out of 27 models. NSGA-II is only able to achieve 100% correctness for 2 models, and produces higher HV for these two models (FM-52 and FM-66).

2- IBEA also achieves significantly better HV in 8 models out of the 27. For the remaining 18 models, NSGA-II achieves better HV in only 2, and the values are close for the rest.

3- NSGA-II takes the lead on the spread indicator, which measures diversity of solutions, for 19 out of 27 models, while IBEA prevails for 5 models, and no winner emerges for the remaining 3 models. But there is no point in comparing HV and spread when %Correct is far below 100%, since HV and spread can have better values due to closeness to optimality in the number of features, defects, used before, and cost, while still having many constraint violations. Hence, there is no significance when HV and spread values achieved by NSGA-II are close to or better than IBEA, while IBEA yields far more valid solutions.

4- In our earlier results (section 6.1), it took IBEA 3 hours to achieve 52% correctness for FM-290 (Electronic Shopping feature model). In this experiment, we are

able to achieve 100% correctness within 5 minutes. This is due to 2 factors: the PUSH technique, and better-tuned parameters. The runtime improvement is *more than* 36-fold (3 hours / 5 minutes). In fact, we will show a much higher runtime improvement in the next section when we add the PULL method.

Table 17 is similar to Table 16, only this time both PUSH and PULL are employed. With PULL, the correctness objective takes 4 times higher priority than each one of the other 4 objectives. Like we did in Table 16, the median value is highlighted in bold if it is statistically better than its counterpart for other algorithms applied to the same feature model, according to the Mann-Whitney test with 95% strength.

The results of Table 17 show the following:

1- The IBEA advantage is maintained, in addition to a significant improvement in %Correct for 3 models, and among them are 2 of the large LVAT models. In fact, we now achieve 100% correctness within the first 5 minutes for all models except FM-88. For that model, %Correct remains at 20% (20 correct configurations out of 100 members in the final population) no matter how long we run IBEA, which tells of an intrinsic complexity and thus a limitation on the number of valid solutions.

2- NSGA-II does not improve when PULL is introduced, since it relies on Boolean dominance criteria which do not account for varying objective weights. The only advantage NSGA-II has here is in the same 2 models as in the previous table, i.e. FM-52 and FM-66.

Table 17: Results of 5-objective optimization; PUSH and PULL

FM	IBEA			NSGA-II		
	%CR	HV	SPRD	%CR	HV	SPRD
FM-43	**100%**	0.31	**0.95**	55%	0.31	0.90
FM43B	**100%**	0.22	0.88	60%	0.22	0.86
FM-44	**100%**	0.25	0.83	35%	0.24	0.85
FM-45	**100%**	0.15	**0.92**	53%	0.15	0.78
FM-46	**100%**	0.24	**1.03**	48%	0.26	0.77
FM-52	100%	0.03	0.74	100%	0.04	**1.01**
FM-53	**100%**	0.22	0.71	64%	0.21	**0.93**
FM-60	**100%**	0.28	0.70	77%	0.27	**0.89**
FM-61	**100%**	0.07	**1.02**	19%	0.11	0.87
FM-63	**100%**	**0.20**	0.70	27%	0.17	0.88
FM-66	100%	0.05	1.03	100%	**0.09**	**1.14**
FM-67	**100%**	0.08	**0.89**	50%	0.08	0.82
FM-70	100%	**0.35**	0.73	84%	0.30	**0.98**
FM-72	**100%**	0.30	0.77	41%	0.28	**0.98**
FM72B	**100%**	0.31	0.74	67%	0.30	**0.97**
FM-88	**20%**	0.16	**1.07**	3%	0.14	0.91
FM-94	**100%**	0.17	0.76	56%	0.17	**0.89**
FM-97	**100%**	0.26	0.66	93%	0.25	**0.96**
FM137	**100%**	**0.31**	0.69	75%	0.28	**0.95**
FM290	**100%**	0.22	0.76	7%	0.20	**1.00**
ToyBox	**100%**	0.22	0.73	4%	0.21	**0.87**
axTLS	**100%**	**0.33**	0.71	3%	0.27	**0.83**
eCos	**100%**	**0.23**	0.84	3%	0.00	0.88
FreeBSD	**100%**	**0.22**	0.96	0%	0.02	0.98
Fiasco	**100%**	0.19	0.81	3%	0.16	**0.87**
uClinux	**100%**	**0.17**	0.74	4%	0.08	0.98
Linux Kernel	0%	0	1.0	0%	0	0.95

3- The HV values in Table 17 cannot be compared to those in Table 16, because of the difference in dimensionality in the objective space. When constraint satisfaction is given 4 times the weight as other objectives, the objective space become 8-dimansional, and HV is calculated as such.

4- In this table, IBEA achieves spread values that are closer to those achieved by NSGA-II. Thus NSGA-II only leads in spread values for 14 models (compared to 19 in Table 16). IBEA leads in 6 models (compared to 5 in Table 6), but also makes better spread values that cancel NSGA-II's lead in the remaining 6 models. This shows that the PULL method not only allows IBEA to achieve correctness faster, but also to achieve more diversity of solutions within the space of valid solutions.

5- The results for the Linux Kernel feature model (6888 features) are not satisfactory, as we do not achieve any correct configurations or any sizeable HV within the first 5 minutes. It is evidenced that we will need a more innovative approach to be able to configure such large model. In [**85**] we achieved a promising result with the Linux Kernel model using a "population seeding" approach. We will expand on this approach in future work and combine it with the PUSH and PULL techniques.

In order to further assess the advantage of the PULL method in improving the performance of IBEA, we chart the median values for TT100% (time to 100%), which is the runtime that it takes IBEA to achieve 100% correctness before and after adding PULL. In Figure 16, the feature models are ordered according to increasing TT100% for IBEA with PUSH but no PULL. FM-88 is missing from the chart because 100% correctness is never achieved for it.

We make the following observations:

1- For the first 17 models in the chart (FM-70 through FM-290), we achieve 100% correctness within 1 second before adding PULL, and stay within 1 second after

adding PULL, although TT100% grows longer with PULL for 12 out of those 17 models. This observation is especially significant for FM-290, for which it took 3 hours to achieve 52% correctness in our original work [**86**] before introducing PUSH or PULL. With both PUSH and PULL, IBEA achieves 100% correctness in 0.665 second; an improvement of over 16,000 times.

2- For the next 5 models (FM-46 through Fiasco), 100% correctness is originally achieved in 3 seconds to 2 minutes. With PULL, TT100% is reduced to between 0.34 and 52 seconds.

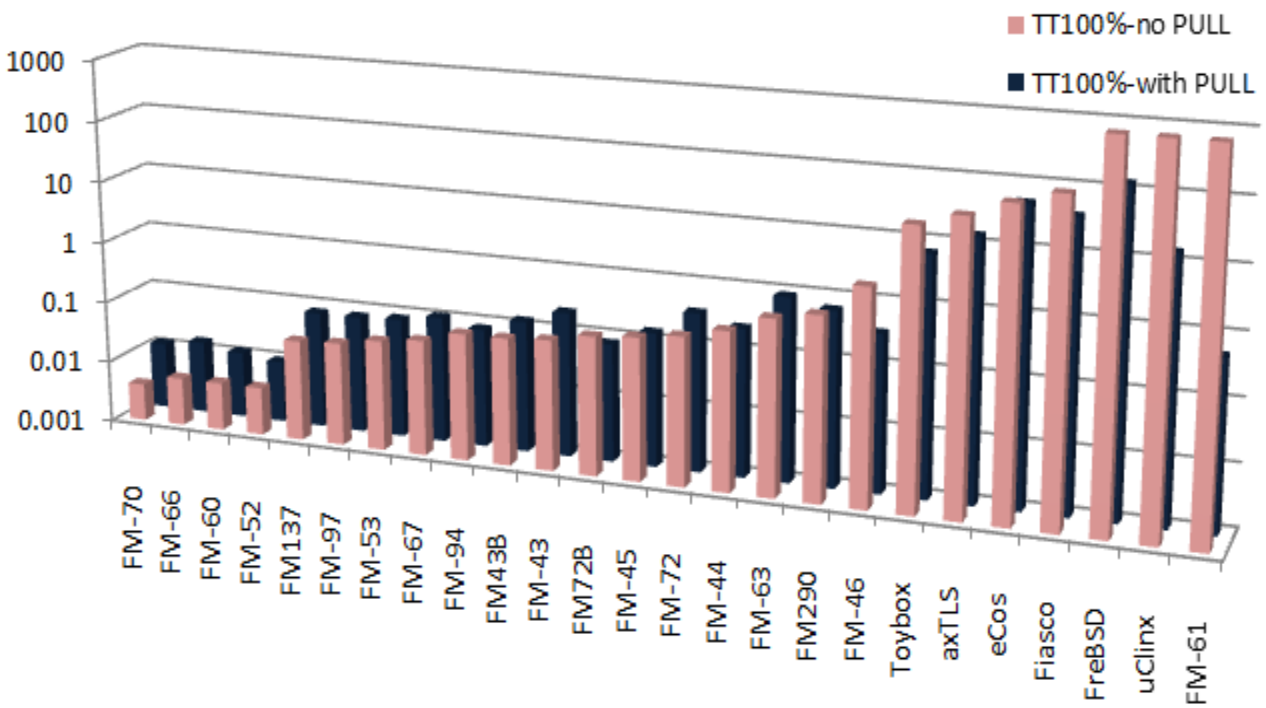

Figure 16: Comparison of TT100% for IBEA with and without the PULL method

3- For the remaining 3 models, 100% correctness is not achieved within the first 5 minutes before PULL (hence the bars extend to 1000 sec). With PULL, IBEA was able to achieve 100% correctness within a median value of 0.52 sec for FM-61, 15 sec for uClinux, and 135 sec for FreeBSD.

6.4 Objective Development over Time

In this section, we show the objective values develop over time as opposed to the development of quality indicators. The objectives, not the quality indicators, represent the user's business concerns and must be the subject of communication with the customer to demonstrate the usefulness of configuration tools.

Figure 17 shows the time development of the quality indicators when applying IBEA to FM-290 (Electronic Shopping) with PUSH and PULL, while Figure 18 shows the time development of the normalized means of objective values. The means are computed for each objective value over all members in the population at each milestone, and normalization is performed in order to visualize all objective values in one graph. The graph shows how the mean number of violations decreases to zero in under a second, and %Correct climbs to 100%. Afterwards, HV climbs until it levels off around the 1-minute mark, indicating that little further optimization is possible beyond that point. Nevertheless, looking at Figure 18, we can see that all objectives bottom out at the 1-minute mark, except for the "Missing Features" objective, which means that the population has reached a milestone in terms of minimizing all the objectives except that

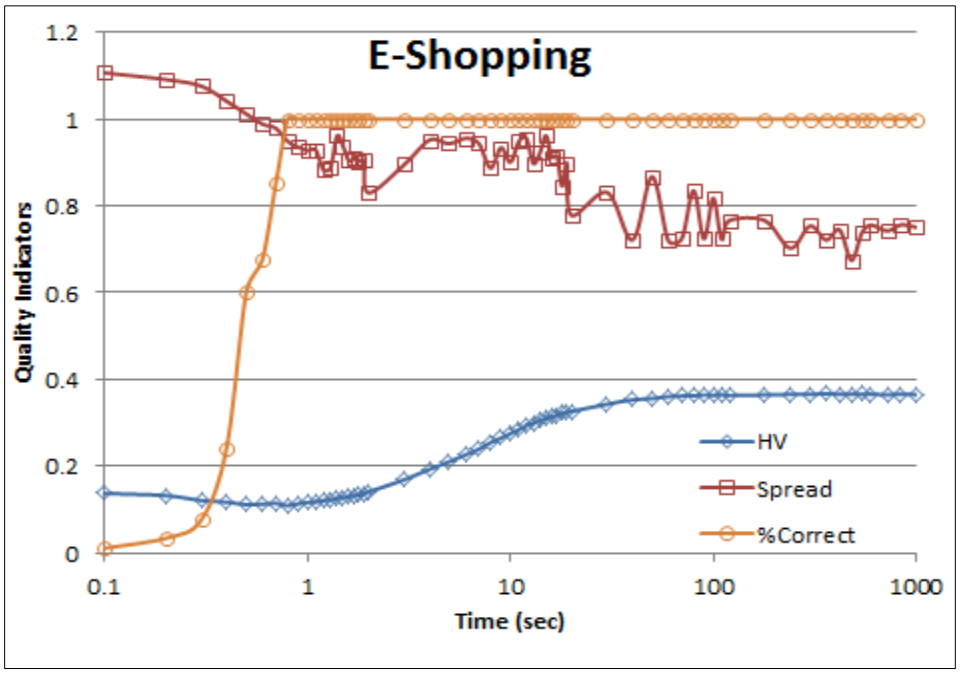

Figure 17: Development of quality indicators over time for E-Shopping

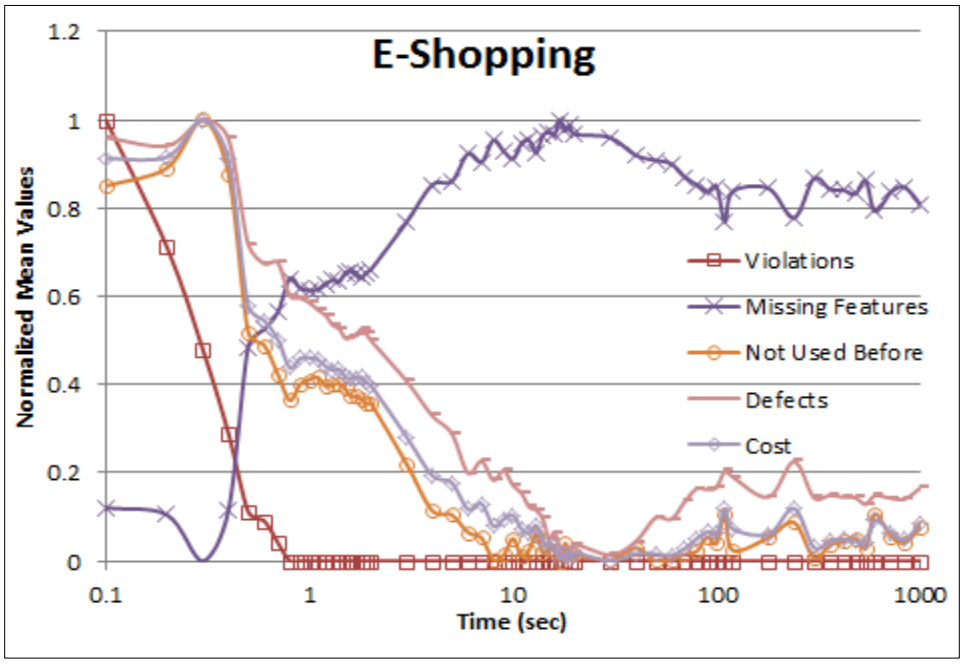

Figure 18: Development of normalized mean values over time for E-Shopping

the number of offered features is also minimized. This means that further optimization would offer more options that push towards increasing offered features, as shown in the development of objective values after the 1-minute mark.

Figures 19 and 20 show the same graphs as Figures 17 and 18, respectively, for IBEA with PUSH and PULL applied to uClinux feature model from the LVAT database. They show that 100% correctness (and zero violations) is achieved at 16 seconds. Afterwards, the population trends toward minimizing the number of missing features, until it bottoms out around the 4-minute mark, after which evolution trends towards less features and thus minimizing the other objectives.

In both cases shown in the Figures 17-20, it can be concluded from observing the mean objective values over time that evolution tends to reach a point after which the objectives continue to play trade-offs against one another. In practice, the user must step in at a certain point and make partial decisions regarding feature selections and acceptable objective values. The optimization tool would then use the user's input and focus the subsequent computation on offering configuration choices that comply with user preferences.

6.5 Population Seeding

The results in section 6.3 show that IBEA was not able to achieve any acceptable configurations within the given 5 minutes for the Linux Kernel feature model composed of 6888 features and about 344,000 rules. We introduce here another strong heuristic to assist IBEA in finding good solutions for the Linux Kernel FM within reasonable time.

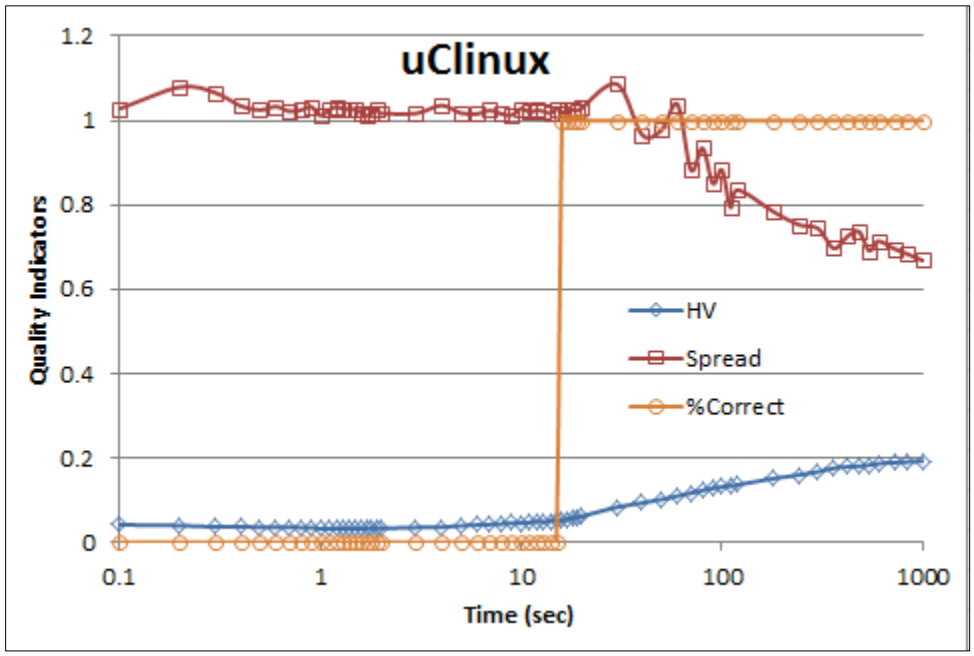

Figure 19: Development of quality indicators over time for uClinux

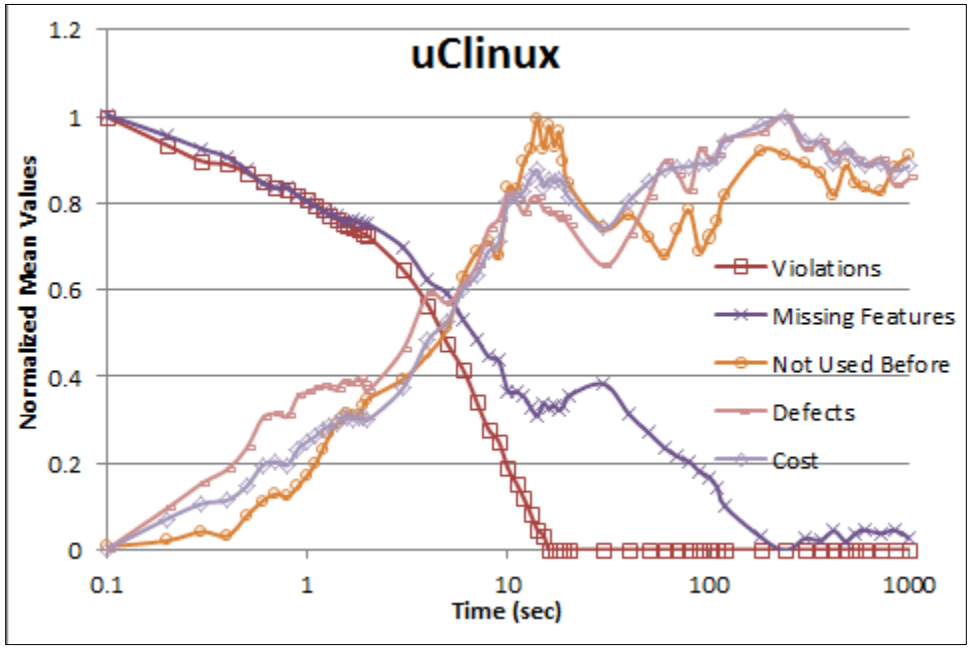

Figure 20: Development of normalized mean values over time for uClinux

Our next strong heuristic technique was to pre-compute a correct configuration and plant it like a seed in the initial population of the evolutionary algorithm. 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.

6.5.1 Parameter Settings

Table 18 shows the parameter values used in this experiment. Notice that a small crossover rate was used in this experiment, although we abandoned crossover altogether in the previous experiment (section 6.3, Table 15). The reason is that crossover is needed to enable the seeding heuristic; otherwise, the population would not be able to learn from the correct seed.

Table 18: Parameter settings for seeding experiment

Parameter	Setting
Population size	300
Archive size	300
Crossover type	Single-Point Crossover
Crossover probability	0.05
Mutation type	Bit-Flip Mutation
Mutation probability	0.001

6.5.2 Method for Generating a Correct Seed

First, we present the way we pre-computed a correct solution 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 a high number of enabled (selected) features, which is more challenging to evolutionary algorithms than finding correct configurations with minimum selected features.

Table 19 shows the time it took this technique to generate a correct solution for each of the 7 LVAT feature models, and the number of selected features within each solution.

Table 19: Time and number of selected features for generated seed

Model	***Total Features***	***Time (sec)***	***Selected Features***
ToyBox	544	10.5	145
axTLS	684	16.5	245
eCos	1244	56	967
FreeBSD	1396	205	946
Fiasco	1638	42	575
uClinux	1850	23	455
Linux X86	6888	11,000 (~3 hours)	5704

6.5.3 Result of Using PUSH and Seeding (no PULL)

The initial result, which appeared in [**85**], shows the result of applying IBEA with feature fixing to the Linux X86 feature model, with the full 5 optimization objectives.

Three different runs were tried separately, for one hour each:

1. No seeding, i.e. all 300 individuals are randomly generated in the initial population.
2. One "feature-rich" seed generated using 2-objective IBEA, along with 299 random solutions.
3. Thirty different fully-correct seeds, generated in a previous run of 5-objective IBEA, along with 270 random solutions.

The number of valid configurations in each of the 3 runs is shown in Figure 21. At the end of 1 hour, the "no seeding" run produced no valid solutions, whereas the two other runs produced about the same number of valid solutions; i.e. 36 solutions due to 1 seed, and 38 solutions due to 30 seeds.

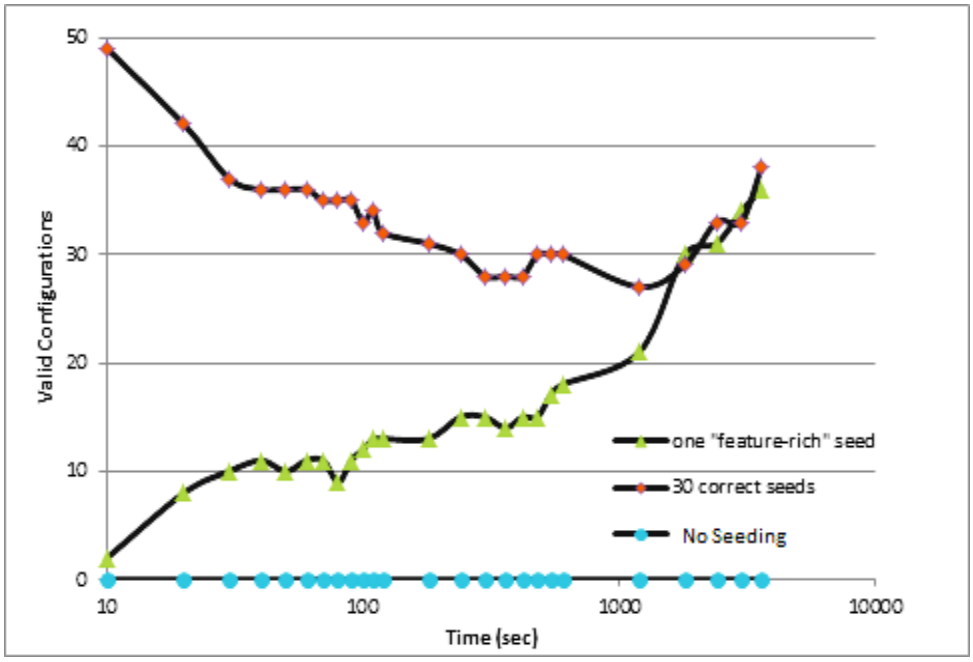

Figure 21: Number of valid configuration due to seeding with PUSH

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 feature-rich seed is compatible with that of 30 seeds.

6.5.4 Result of Using PUSH, PULL, and Seeding

Next we show the result of adding the PULL heuristic to PUSH and seeding (with 1 seed only).

Figure 22 shows the number of correct configurations (out of 300 candidates in each generation) due to each crossover rate, over the course of 10 hours of 5-objective optimization. We observe that the number of valid configurations increases more rapidly than in Figure 21; and by the 1-hour mark we achieve 167 solutions, compare with 36 valid configurations achieved when PULL is not employed. This clearly shows the advantage of the PULL heuristic in promoting conformance with the feature model.

We also notice that the number of valid configurations begins to level off after 3 hours, when it reaches 250 solutions (about 83% of the population.) At the end of 10 hours, the number goes up to 282 solutions (94% of the population.)

Next, we present the result of a larger experiment involving all 7 feature models from the LVAT repository, and focusing on the overall quality indicators for resulting Pareto Front after 5 minutes of 5-objective optimization. The results are shown in Table 20, where the median values of %Correct, HV, and Spread are reported for 10 indepen-

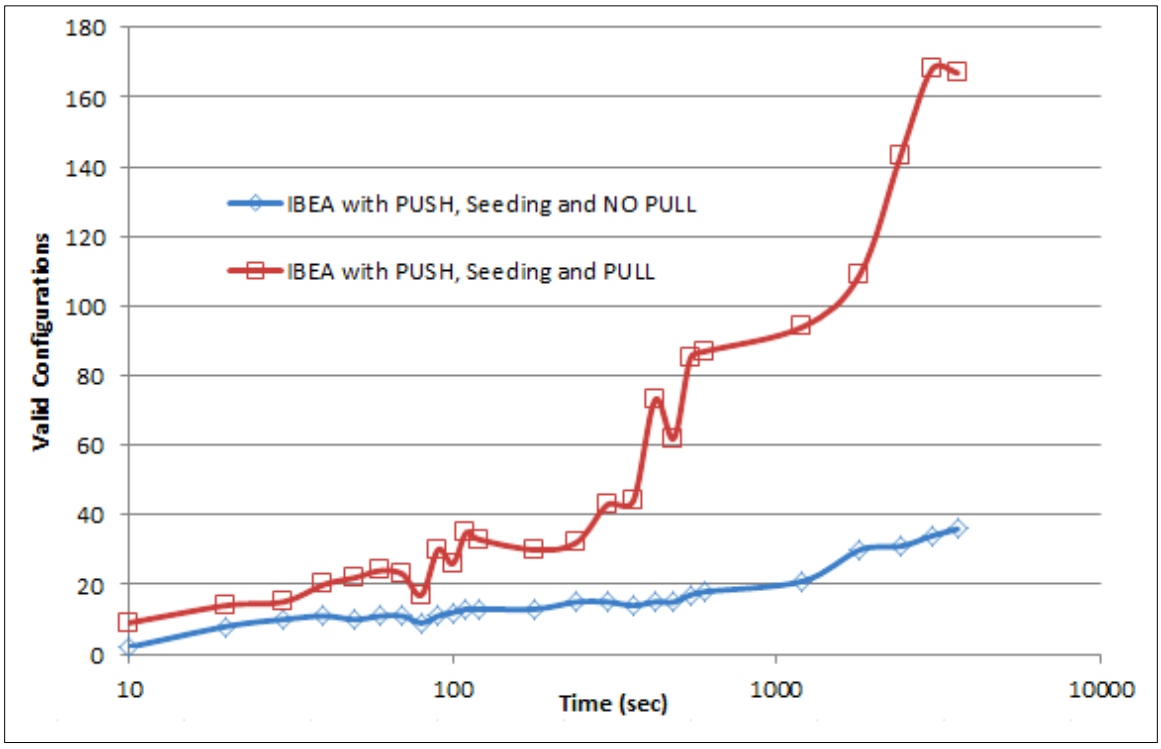

Figure 22: Number of valid configuration due to seeding with PUSH and PULL

Table 20: Results of 5-objective optimization; PUSH, PULL, and Seeding

Feature Model	IBEA with PUSH, PULL, And NO Seeding (Results from Table 17)			IBEA with PUSH, PULL, And Seeding		
	%CR	**HV**	**SPRD**	**%CR**	**HV**	**SPRD**
ToyBox	100%	0.22	0.73	100%	0.21	0.74
axTLS	100%	0.33	0.71	100%	0.33	0.73
eCos	100%	0.23	0.84	100%	0.23	0.84
FreeBSD	100%	0.22	0.96	100%	0.22	0.96
Fiasco	100%	0.19	0.81	100%	0.20	0.77
uClinux	100%	0.17	0.74	100%	0.18	0.70
Linux Kernel	0%	0	1.0	**43%**	**0.0015**	1.01

dent runs with Seeding, and compared with previous results from Table 17.

We make the following observations:

1- The overall HV and Spread are the same or close with or without seeding. The only exception is the Linux Kernel feature model, for which a nonzero HV is achieved for the first time since we are now able to find valid configurations and optimize around them.

2- We still achieve 100% population without any violations for the first 6 feature models. The Linux Kernel model now achieves 43% of its final 300-member population as valid configurations, which is a large and significant achievement.

6.5.5 Investigating the Diversity of Solutions Achieved with Seeding

The seeding technique is expected to produce solutions that are close together, since the population is "infected" with one strong seed that gets implanted in the initial generation. In order to see how closely the resulting configurations resemble the seed, we look at box plots of the objective values for those correct configurations achieved at each time milestone.

Figure 23 shows the box plot against time for the number of selected features when examining the valid configurations alone. Figure 24 shows the box plot against time for the features used before. Figure 25 shows the box plot against time for the total number of known defects. Figure 26 shows the box plot against time for the total cost.

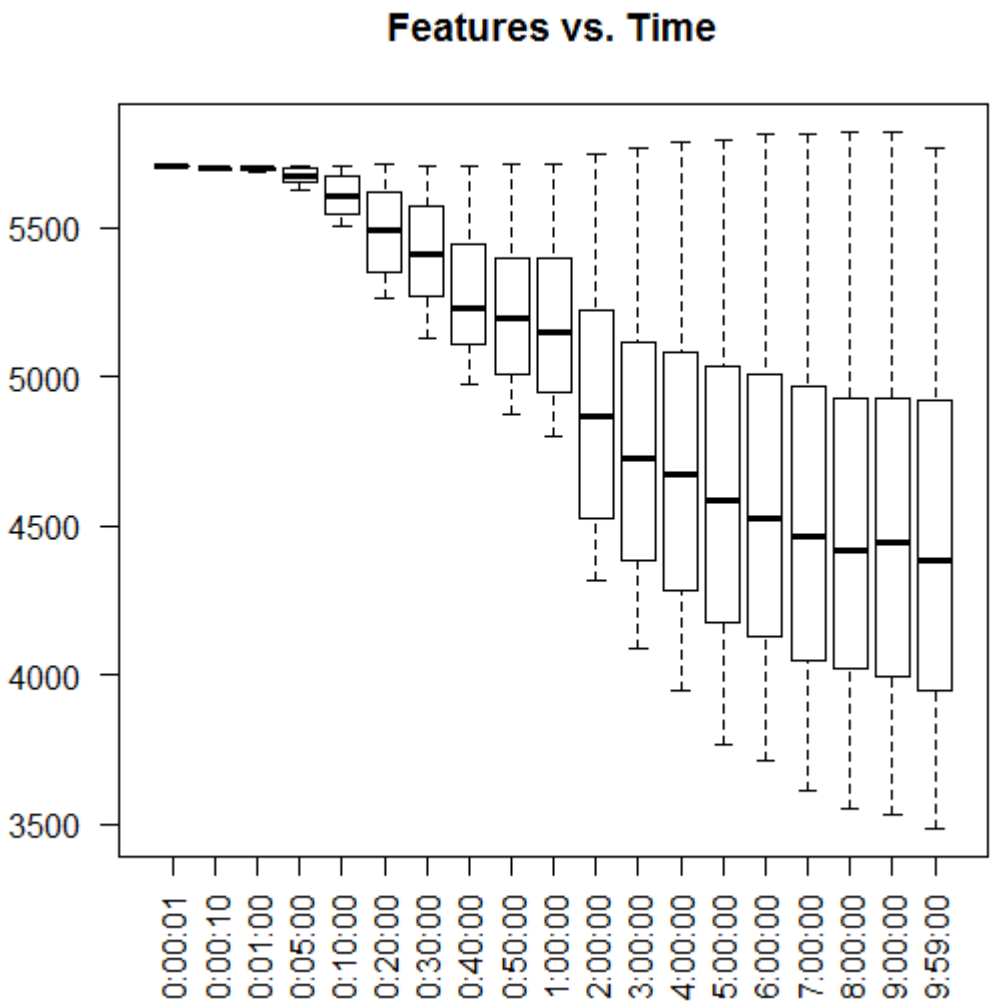

Figure 23: Box plot against time for the number of selected features.

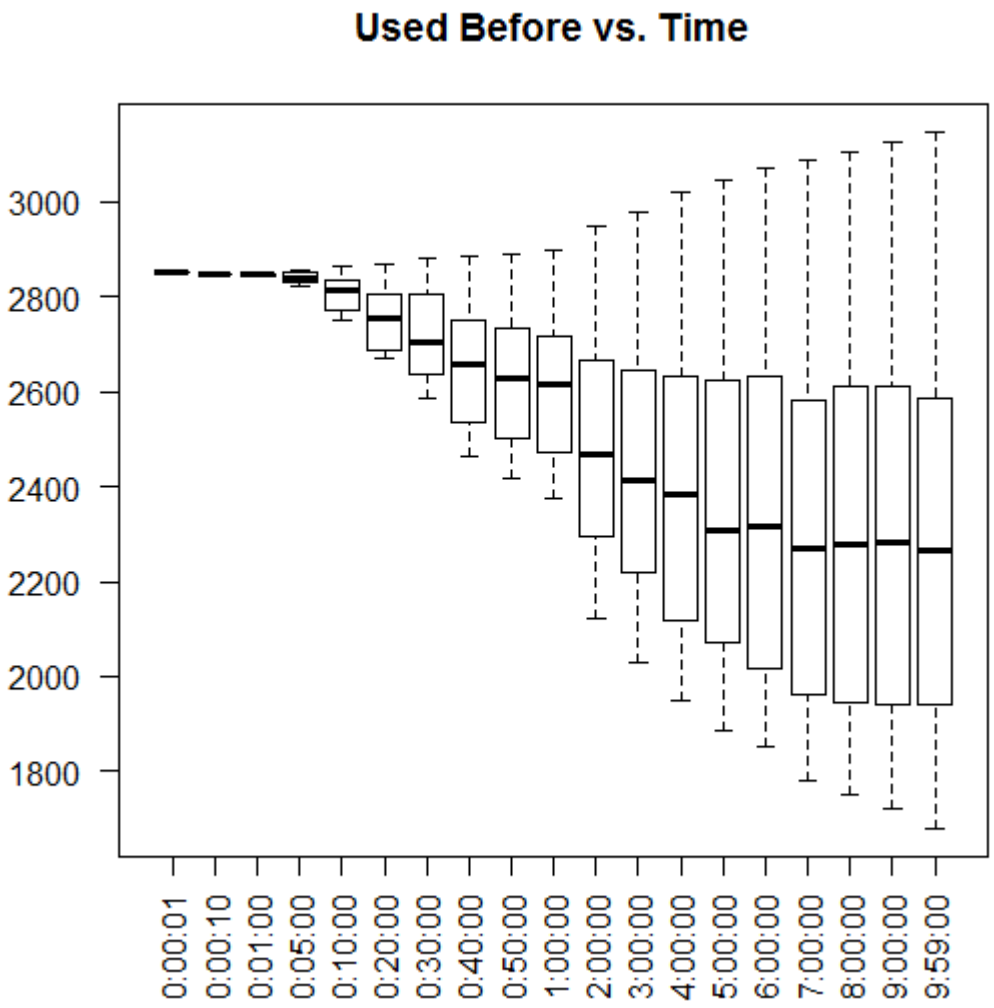

Figure 24: Box plot against time for the number of features used before.

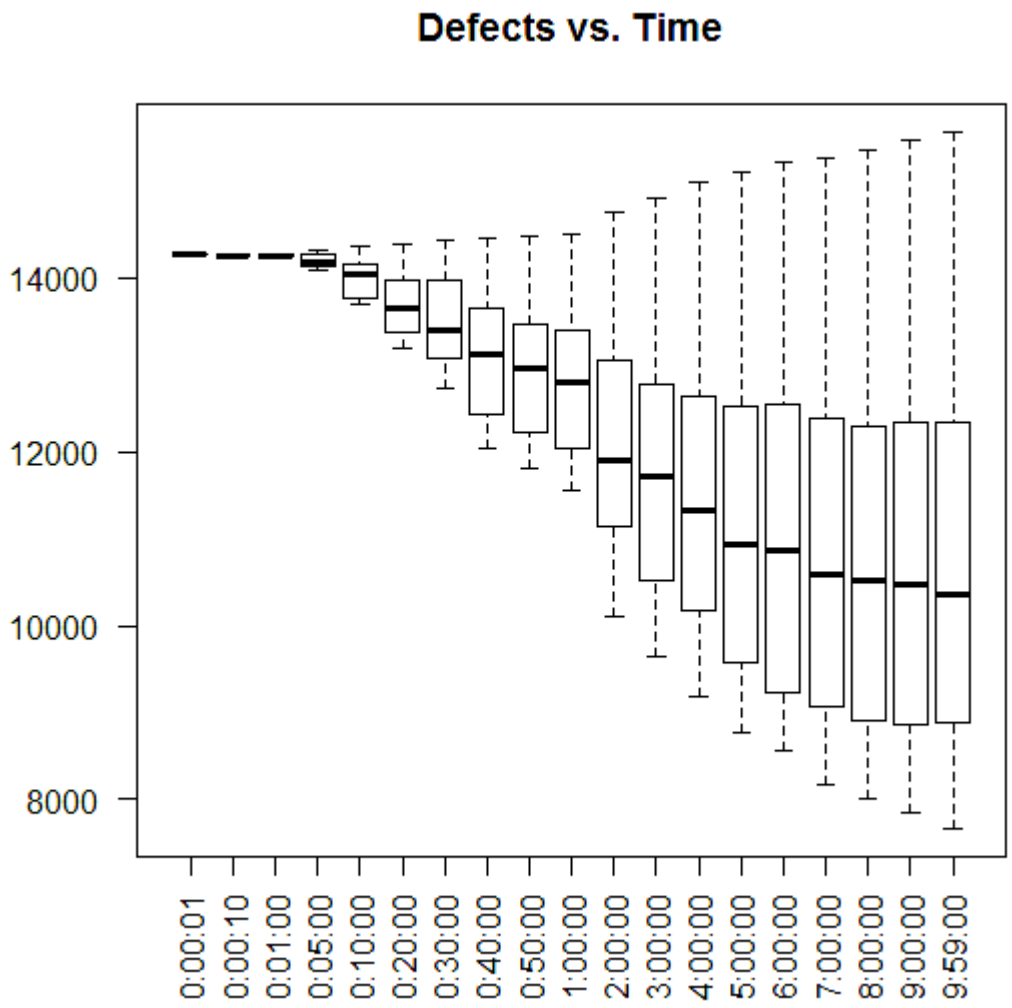

Figure 25: Box plot against time for the number of known defects.

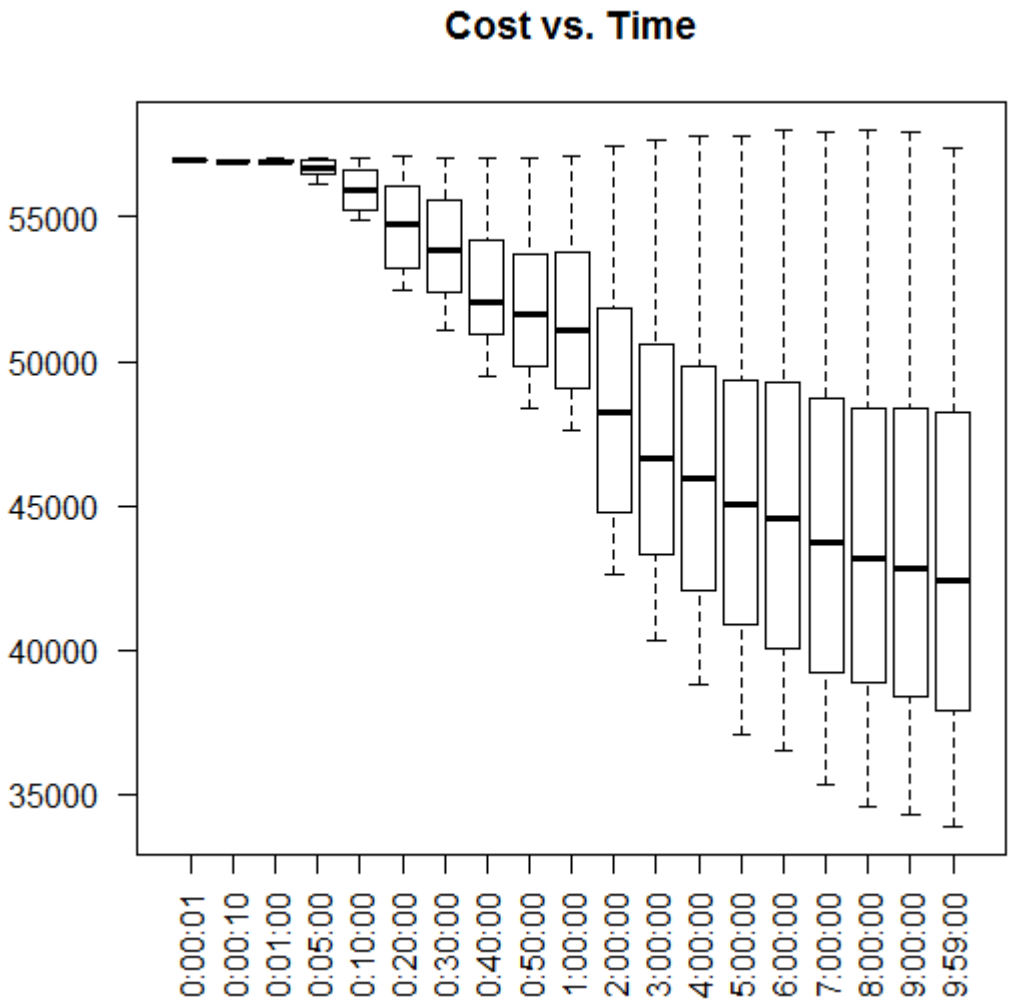

Figure 26: Box plot against time for the total cost.

We make the following observations regarding the box plots in Figures 23-26:

1- The valid configurations are strongly influenced by the seed, which is represented by the single valid configuration found at 1 second. This is evident in the way the

objective values develop slowly afterwards and do not deviate far away from the objective values for the seed.

2- The evolutionary process slows down after the first few hours, which suggests that running the algorithms for longer than 10 hours is less likely to result in finding new and higher optimized configurations.

3- Nevertheless, this observation can be exploited in the following manner: suppose we obtain prior knowledge of the user's preferences in terms of a desired maximum cost or defect that they can tolerate; if we compute a seed that falls within that preference area we can use it to bias the search in the 5-objective space towards optimizing solutions within a narrow area that is of most interest to the user, rather than present a Pareto Front that is far stretched across an ocean of possible configurations. This idea is left for future work.

Chapter 7: Threats to Validity

Validity of research results has four components: construct, internal, conclusion, and external validity.

7.1 Threats to Construct Validity

Construct validity refers to the extent to which a piece of research actually investigates what the researcher purports to investigate. An examples of threat to construct validity is the confounding factors that are not considered in the study. In our study, possible factors that may affect construct validity are:

7.1.1 Sampling Bias

Sampling bias refers to the choice of subject data for the experiments. We used 20 feature models from SPLOT and 7 from LVAT. SPLOT has hundreds of feature models, but most of them are small and simple, not posing much challenge to the problem of optimum feature selection. For example, SPLOT as of today (April 2014) hosts 481 feature models, the smallest containing 10 features, while the largest having 366 features. Out of 481 models, only 7 models have more than 100 features. We chose 20 feature model that varied in size from medium (43 features) to large (290 features—the largest in SPLOT until lately.) We also made sure to exclude feature models that did not include any cross-tree constraints, because we believed they were far removed from the reality of software product lines. From LVAT, which is host to 15 very large feature models that were reverse-engineered from Linux software packages, we chose 7 models going up to 6888 features, which was the most challenging to our tools. We excluded larger feature

models because they did not match the published data about the underlying software packages.

Another subject bias 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 numbers 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 continuous dominance (IBEA) over Boolean dominance algorithms which couldn't possibly be biased by the synthetic data. Future work should attempt to collect real data for use with other MOEAs to optimize product configuration.

7.1.2 Treatment Bias

Treatment Bias refers to the limited set of treatments that are applied and compared in the experiments, since questions might arise about other treatments and how they might have performed when applied to the experiment subjects. Many algorithms exist in the world of multi-objective optimization, and many fall in the category of MOEAs. The "Algorithm" columns in Tables 2-5 of this study show the variety of MOEAs applied in search-based software engineering (SBSE). When we started experimenting with feature models, we tried to include every algorithm that was available to us, but it quickly became clear that any algorithm that is based on Boolean dominance was yielding the same poor results when the number of objectives is increased (Tables 10

and 11) and with larger feature models. The continuous dominance algorithm (IBEA) was the only one that gave satisfactory results. This was the reason why we limited our subsequent experiments to comparing IBEA with NSGA-II, the most widely-used of Boolean dominance MOEA. Nevertheless, we welcome our colleagues in SBSE to try different algorithms to this domain and others, in order to confirm, refute or improve the results we achieved with continuous dominance.

7.1.3 Parameter Bias

Back in section 6.2, we proved the importance of parameter tuning in SBSE, as did others before us [**3**]. We also showed that continuous dominance (e.g. IBEA) was still the optimizer of choice no matter what parameters were used. Therefore, we did not perform the same type of rigorous parameter tuning for the remaining experiments, rather we made parameter choices based on experience. We acknowledge that the best practice to account for the variability of performance with parameters is adaptive parameter control, where the parameters are changed dynamically according to the performance they yield in each generation in the evolutionary process.

7.1.4 Measurement Bias

Measurement bias refers to the metrics we use to assess which methods are better than others. The essence of multi-objective optimization is not being able to use a single objective (or an aggregation of objectives into one metric) for assessing the final solution. Rather, the resulting Pareto Front as a whole is assessed using quality measures such as the hypervolume (HV), Spread, Generational Distance (GD), etc. The two indicators that

we used are representative of the qualities that are sought in a Pareto Front; HV measures closeness to optimality, while Spread measures the diversity of solutions. We added the %Correct measure to assess how many solutions in the Pareto Front completely conform to the rules in the feature model, which is an essential indicator of success in product configuration. In addition, we made sure to look at the development of individual; objectives over time (Figures 17-20, and Figures 23-26) in order to show how the user's preferences get optimized as they compete against each other. Nevertheless, the choice of quality indicators and success measures when dealing with a Pareto Front remains an open question worthy of further examination.

7.2 Threats to Internal Validity

Internal validity is concerned with influences that may affect the independent variables or measurements without the researcher's knowledge. One possible threat to internal validity in our study is the fact that we relied on jMetal's implementation of the MOEAs, since efficiency can vary among different implementations. Furthermore, the MOEAs were implemented in Java with its garbage collection utility, which lies outside of any programmer's control. Nevertheless, our use of multiple runs and statistical testing should alleviate this threat and assure the strength of conclusions.

7.3 Threats to Conclusion Validity

In comparing the performance of MOEAs, we performed statistical significance testing (Mann-Whitney U test) on 10 different runs of each MOEA. The goal was to

remove the randomized nature of the algorithms as a confounding factor. It may be argued that 10 runs are not enough for a strong conclusion, but the high significance achieved in the tests (≥ 95% strength) supports our conclusions.

7.4 Threats to External Validity

A threat to external validity is that we are unable to generalize our findings to other software engineering problems, since the scope of any experiment needs to be defined for practical reasons. Nevertheless, we do provide a discussion of the problem characteristics that make continuous dominance (IBEA) perform best for software feature models, and we anticipate the same performance advantage when applying IBEA to problems with high complexity and dimensionality, because of the way continuous dominance makes use of all the knowledge in the objective values that the user cares about. In order to assess this claim, we invite researchers in SBSE to consider continuous dominance algorithms (e.g. IBEA) when choosing MOEAs for multi-objective problems.

Chapter 8: Discussion

In this chapter, we reason about the findings of this paper and their implications.

8.1 Method for Achieving Correct Configurations

In this work, we were able to generate many acceptable configurations with the help of evolutionary algorithms via defining correctness as one of the optimization objectives. Other researchers [**10**] included a repair operator to guarantee that each candidate solution conformed with the feature model before evaluating it for promotion to the next generation. They refrained from designing the objective function so that invalid solutions always score lower than valid solutions, because it was hard to do so with a single-objective fitness function. Having attempted the inclusion of correctness within another objective in order to guarantee correctness, we recognize this hardship. It was much easier for us to obtain correct solutions (especially with IBEA) by defining correctness as an objective of its own.

Furthermore, we were able to accelerate the search for correct configurations by restricting the evolutionary operators to the dependencies defined in the tree structure of the feature models (the PUSH method). Additionally, the PULL technique gave more weight to minimizing rule violations in the optimization process; improving performance.

8.2 Indicator-Based Evolutionary Algorithm (IBEA) vs. Other MOEAs

IBEA performs much better than commonly used algorithms because of the difference in fitness ranking criteria which was explained in section 2.7. In [**98**], 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 like ε-MOEA, MSOPS, IBEA and SMS-EMOA cope very well with high-dimensional objective spaces. It is argued that NSGA-II and SPEA2 tend 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." Those results were obtained with nonlinear function solving. Our work verified the same results.

All other algorithms used in this study (except IBEA) depend on evaluation criteria similar to NSGA-II or SPEA2, thus inheriting the same handicap at higher dimension; they tended to prune out solutions that crowded towards the much-desired zero-violation point, thus achieving low scores on the %Correct measure.

IBEA, on the other hand, calculates a dominance value based on quality indicators that depend primarily on the user preferences. Thus IBEA has no need for a secondary evaluation criterion such as diversity of solutions. This enabled IBEA to explore the various optimal solutions with zero-violations of model constraints.

This actually highlights IBEA's ability to find correct as well as optimal solutions. Other algorithms focused on finding optimal and diversified solutions, but failed in both, because they couldn't explore the fully-correct solution space. Evolutionary search algorithms are not only meant for optimization, although they're mostly used for it, but

they're also meant to find feasible solutions from a large space of possible combinations that may or may not conform to model constraints and dependencies.

8.3 Run Time Improvements

We were able to achieve huge run time savings (about 16,000-fold improvement) over the initial results by restricting the mutation operator to cease from violating the feature model dependencies expressed in the feature tree (the PUSH method). This, along with no crossover and a low rate of mutation, exploited the problem structure to accelerate the search through the space of possible configurations. Such customization is crucial to the success in addressing any type of problem, as the evolutionary algorithms are generic approaches to solving problems, and may not fit the nature of the search space.

8.4 Potential for "User-In-The-Loop" Design

As demonstrated in Fig. 9, we were able to obtain "acceptable" i.e. model-conforming results within 1 second for medium-sized feature models from SPLOT. The large real-life feature models from LVAT took longer to achieve an all-correct population, but the longest time was 135 seconds for the FreeBSD feature model. This empowers us to explore the idea of interactive configuration, with the user being part of the optimization loop. A precondition for including the user in the loop is the ability to provide the user with meaningful candidate solutions within a short period of time, i.e. to offer advice before an expert's attention wanders to other issues. Neilson [**72**] reports that

“the basic advice regarding response times has been about the same for thirty years; i.e.

- 1 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay
- 10 seconds is the limit for keeping a user's attention focused on the dialogue."

In the case of aggregate-based approaches (single optimal solution), the user is consulted only once at the beginning of the process, their preferences are taken as input, the weights for the objectives are preset, then an optimum one-of-a-kind output is presented in the end. Whereas in the case of Pareto-based approaches (range of optimal solutions), the user is consulted in the beginning, and then again in the end when they need to choose from among the Pareto-optimal solutions. This gives the users insight and more decision-making power.

Interactive approaches (or so-called preference-based approaches) provide the users with an even bigger role in the optimization process; they are consulted several times (typically) before, during, and after the machines compute the optimum solutions. This is especially useful when the underlying optimizer is an evolutionary algorithm, which typically evolves thousands of candidate solutions in every possible direction throughout the decision space, and goes through the evaluation and ranking of every possible candidate. Such lengthy process can be cut short if the user is able to limit the search to a certain region of interest within the decision space or the objective space, or both. Examples of preference-based optimization algorithms are [**92**] and [**21**].

Chapter 9: Conclusions and Future Work

9.1 Conclusions

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. [**46**]

Our experience, as reported in this study, 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 (PUSH, PULL, and population seeding) was our key to scalability. One feature-rich valid seed in the midst of a 300-member initial population was enough to generate many valid configurations within a short period. 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”.

We have also shown a four-orders-of-magnitude improvement in IBEA's runtime when handling the E-Shopping feature model as compared to initial results. This remarkable result was possible due to respecting the internal structure of the feature models (PUSH), giving priority to constraint satisfaction (PULL), and better parameter tuning. We also achieved scalability of the remarkable results obtained with IBEA with larger feature models, such as the Linux kernel feature model (part of LVAT repository) composed of 6888 features. The "population seeding" approach guided IBEA into finding valid configurations early in the optimization process.

9.2 Future Work

Broadly speaking, there needs to be a better understanding of the decision making process and what it requires in various disciplines, where the decision makers must be provided with the most comprehensive, yet concise, picture about the trade-offs involved with complex decision spaces, and the interactions of various decision points.

Decision making can be a complex task in various fields, such as government planning, electric power systems, or large-scale software design. Complexity may occur in the decision space or in the objective space, or both. The decision space can have many intricate constraints and dependencies among individual decisions such that finding logically-sound solutions becomes as tough a problem as finding optimal solutions. The objective space can have multiple conflicting objectives such that optimizing a certain objective would almost certainly be detrimental to other objectives. When complexity occurs in both the decision and objective spaces, automated decision support tools

become indispensable to the decision maker, especially with far-reaching consequences of strategic decision making in the present age.

The state of the art in complex decision making in various fields is using Multi-Objective Evolutionary Algorithms (MOEAs), with their ability to find a range of Pareto-efficient solutions, i.e. a group of optimal solutions where each solution represents the best trade-off among various competing objectives. This approach has eclipsed traditional reasoning approaches which become inefficient and time-consuming when dealing with complex systems. Furthermore, it would be a disservice to the decision maker if all the multiple objectives are lumped together in one formula with pre-determined weights, and then presenting a single optimal solution as a result. Such practice only hides the richness of the decision space and the enlightening knowledge in the objective space; thus leaving the decision makers with incomplete data and sub-optimal reasoning.

In each of the aforementioned fields (government planning, electric power systems, and large-scale software design), multi-objective evolutionary algorithms are used by many researchers to solve various problems, with two recurring tendencies:

First, the objective space was often reduced into two or three competing objectives, such that the resulting range of solutions could nicely be plotted in 2D or 3D.

Second, the algorithms chosen to perform the optimization put much emphasis on finding a diverse range of solutions that the user preferences did not play a central role in the multi-objective search.

In this dissertation, we have arrived at two major findings: if your system is complex in both the decision and objective spaces, preserve its complexity by expressing all the available user preferences without aggregation, and use an optimization algorithm that exploits user preferences in multi-objective search. The results obtained with this approach will be far-more optimal and diversified than those obtained with the traditional algorithms that are less sensitive to user preferences. The other major finding was the importance of customizing the evolutionary search according to the nature of the decision space at hand. Much search time can be saved by tailoring the evolutionary operators to abide by the domain constraints.

We envision a far reaching impact of these results; in spite of the previous research work that dealt with multi-objective optimization in various fields, the real power within these algorithms has not yet been unleashed. The suggested direction of research is: to preserve complexity in system modeling; to involve more and more data that expresses user/decision maker preferences; and to deploy algorithms that best utilize user preferences in the quest for optimization. Furthermore, the various trade-offs need to be presented to the decision makers in an intuitive and insightful manner that empowers them to make the best decisions possible according to the available data.

Directions for future work may include:

1. Improving the configuration process by including the user-in-the-loop approach, where the optimizer runs for a few seconds, presents a range of results to the user, and then takes the user's choices as input to the next round of optimization. This

trend was followed by prominent researchers in the field of MOEAs such as Thiele et al. [**92**], and Chaudhuri and Deb [**21**].

2. Exploring the adaptive parameter control approach where multiple populations are evolved using different sets of parameters; their performance being reported to a central manager, and parameters are dynamically controlled for new generations following a performance assessment.

References

[1] Henrik Reif Andersen, "An Introduction to Binary Decision Diagrams," The IT University of Copenhagen, Lecture notes for Efficient Algorithms and Programs 1999.

[2] A. Arcuri and G. Fraser, "On Parameter Tuning in Search Based Software Engineering," in *Proc. SSBSE*, 2011, pp. 33-47.

[3] A. Arcuri and G. Fraser, "Parameter Tuning or Default Values? An Empirical Investigation in Search-Based Software Engineering," *Empirical Software Engineering*, February 2013.

[4] A. Arcuri, D. R. White, J. Clark, and X. Yao, "Multi-Objective Improvement of Software using Co-Evolution and Smart Seeding," in *Proc. SEAL*, Melbourne, Australia, 2008, pp. 61-70.

[5] W. K. G. Assunção, T. E. Colanzi, A. T. R. Pozo, and S. R. Vergilio, "Establishing Integration Test Orders of Classes with Several Coupling Measures," in *Proc. GECCO*, Dublin, Ireland, 2011, pp. 1867-1874.

[6] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani, "Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features," in *Proc. SPLC*, Jeju Island, South Korea, 2010, pp. 300-315.

[7] M. O. Barros, "An Analysis of the Effects of Composite Objectives in Multiobjective Software Module Clustering," in *Proc. GECCO*, Philadelphia, USA, 2012, pp. 1205-1212.

[8] Don Batory, "Feature Models, Grammars, and Propositional Formulas," in *Proceedings of the 9th International Software Product Lines Conference*, Rennes, France, 2005, pp. 7-20.

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

[10] D. Benavides, A. Ruiz-Cortés, and P. Trinidad, "Coping with automated reasoning on software product lines," in *Proc. 2nd Groningen Workshop on Software Variability Management*, 2004.

[11] 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.

[12] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "A Study of Variability Models and Languages in the Systems Software Domain," *IEEE TSE*,

2013.

[13] 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

[14] 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.

[15] 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

[16] B. Boehm, "Value-Based Software Engineering," *Software Engineering Notes*, vol. 28, no. 2, pp. 1-12, March 2003.

[17] M. Bowman, L.C. Briand, and Y. Labiche, "Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 817-837, 2010.

[18] M. M. A. Brasil, da Silva T. G. N., F. G. de Freitas, J. T. de Souza, and M. I. Cortés, "A Multiobjective Optimization Approach to the Software Release Planning with Undefined Number of Releases and Interdependent Requirements," in *Proc. ICEIS*, Beijing, China, 2011.

[19] R. Britto, P. S. Neto, R. Rabelo, W. Ayala, and T. Soares, "A Hybrid Approach to Solve the Agile Team Allocation Problem," in *Proc. CEC*, Brisbane, Australia, 2012, pp. 1-8.

[20] R. da Veiga Cabral, A. T. R. Pozo, and S. R. Vergilio, "A Pareto Ant Colony Algorithm applied to the Class Integration and Test Order Problem," in *Proc. ICTSS*, Natal, Brazil, 2010, pp. 16-29.

[21] S. Chaudhuri and K. Deb, "An Interactive Evolutionary Multi-Objective Optimization and Decision Making Procedure," *Applied Soft Computing*, vol. 10, pp. 496–511, 2010.

[22] F. Chicano, F. Luna, A. J. Nebro, and E. Alba, "Using Multi-objective Metaheuristics to Solve the Software Project Scheduling Problem," in *Proc. GECCO*, Dublin, Ireland, 2011, pp. 1915-1922.

[23] T. E. Colanzi, W. Assuncao, S. R. Vergilio, and A. Pozo, "Generating Integration Test Orders for Aspect-Oriented Software with Multi-objective Algorithms," in *Latin American Workshop on Aspect-Oriented Software Development*, 2011.

[24] T. E. Colanzi and S. R. Vergilio, "Applying Search Based Optimization to

Software Product Line Architectures: Lessons Learned," in *Proc. SSBSE*, Riva del Garda, Italy, 2012, pp. 259–266.

[25] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the ACM Symposium on Theory of Computing*, New York, NY, USA, 1971, pp. 151-158.

[26] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software*, vol. 15, no. 6, pp. 37-45, 1998.

[27] M. Cordy, P. Y. Schobbens, P. Heymans, and A. Legay, "Beyond boolean product-line model checking: dealing with feature attributes and multi-features," in *Proc. ICSE*, San Francisco, USA, 2013, pp. 472-481.

[28] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing Cardinality-Based Feature Models and Their Specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7-29, 2005.

[29] K. Czarnecki and A. Wasowski, "Feature Diagrams and Logics: There and Back Again," in *Proc. SPLC*, Kyoto, Japan, 2007, pp. 23-32.

[30] F. G. de Freitas and J. T. de Souza, "Ten Years of Search Based Software Engineering: A Bibliometric Analysis," in *Proc. SSBSE*, Szeged, Hungary, 2011, pp. 18-32.

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

[32] J. T. de Souza, C. L. Maia, F. G. de Freitas, and D. P. Coutinho, "The human competitiveness of search based software engineering," in *Proc. SSBSE*, Benevento, Italy, 2010, pp. 143-152.

[33] 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.

[34] 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.

[35] J. J. Durillo, Y. Zhang, E. Alba, M. Harman, and A. J. Nebro, "A Study of the Bi-Objective Next Release Problem," *Empirical Software Engineering*, vol. 16, no. 1, pp. 29-60, February 2011.

[36] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro, "A Study of the Multi-Objective Next Release Problem," in *Proc. SSBSE*, Windsor, UK, 2009, pp. 49-58.

[37] A.E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*.: Springer, 2003.

[38] H. Eskandari, C. D. Geiger, and G. B. Lamont, "FastPGA: A Dynamic Population Sizing Approach for Solving Expensive Multiobjective Optimization Problems," in *Proceedings of EMO*, 2007, pp. 141-155.

[39] J. Ferrer, F. Chicano, and E. Alba, "Evolutionary Algorithms for the Multi-objective Test Data Generation Problem," *Software: Practice and Experience*, vol. 42, pp. 1331–1362, 2012.

[40] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, "Not Going to Take this Anymore: Multi-Objective Overtime Planning for Software Engineering Projects," in *Proc. ICSE*, San Francisco, USA, 2013.

[41] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang, "“Fairness Analysis” in Requirements Assignments," in *Proc. RE*, 2008, pp. 115-124.

[42] S. Frey, F. Fittkau, and W. Hasselbring, "Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud," in *Proc. ICSE*, San Francisco, USA, 2013.

[43] L. Grunske, "Identifying "Good" Architectural Design Alternatives with Multi-Objective Optimization Strategies," in *Proc. ICSE*, Shanghai, China, 2006, pp. 849-852.

[44] S. Gueorguiev, M. Harman, and G. Antoniol, "Software Project Planning for Robustness and Completion Time in the Presence of Uncertainty using Multi Objective Search Based Software Engineering," in *Proc. GECCO*, 2009, pp. 1673-1680.

[45] 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.

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

[47] M. Harman, "The Current State and Future of Search Based Software Engineering," in *Proc. FOSE*, 2007, pp. 342-357.

[48] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, December 2001.

[49] M. Harman, K. Lakhotia, and P. McMinn, "A Multi-Objective Approach to Search-based Test Data Generation," in *Proc. of GECCO*, London, UK, 2007, pp. 1098–1105.

[50] 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.

[51] M. Harman, , P. McMinn, J. T. de Souza, and S. Yoo, "Search Based Software Engineering: Techniques, Taxonomy, Tutorial," *Empirical Software Engineering and Verification*, vol. 7007, pp. 1-59, 2012.

[52] M. Harman and L. Tratt, "Pareto Optimal Search Based Refactoring at the Design

Level," in *Proc. GECCO*, London, England, 2007, pp. 1106-1113.

[53] W. Heaven and E. Letier, "Simulating and Optimising Design Decisions in Quantitative Goal Models," in *Proc. RE*, 2011, pp. 79-88.

[54] 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.

[55] 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.

[56] K. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, vol. 19, no. 4, pp. 58-65, Jul/Aug 2002.

[57] T. M. Khoshgoftaar and Y. Liu, "A Multi-Objective Software Quality Classification Model Using Genetic Programming," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 237-245, June 2007.

[58] T. M. Khoshgoftaar, Y. Liu, and N. Seliya, "A Multiobjective Module-Order Model for Software Quality Enhancement," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 6, pp. 593-608, December 2004.

[59] T. Kremmel, J. Kubalik, and S. Biffl, "Software Project Portfolio Optimization with Advanced Multiobjective Evolutionary Algorithms," *Applied Soft Computing*, vol. 11, no. 1, pp. 1416-1426, January 2011.

[60] A.C. Kumari, K. Srinivas, and M.P. Gupta, "Software Requirements Selection using Quantum-inspired Elitist Multi-objective Evolutionary Algorithm," in *Proc. ICAESM*, India, 2012, pp. 782-787.

[61] W. B. Langdon, Harman M., and Jia Y., "Efficient Multi Objective Higher Order Mutation Testing with Genetic Programming," *Journal of Systems and Software*, vol. 83, no. 12, pp. 2416-2430, December 2010.

[62] 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.

[63] Z. Liu, H. Guo, D. Li, T. Han, and J. Zhang, "Solving Multi-objective and Fuzzy Multi-attributive Integrated Technique for QoS-Aware Web Service Selection," in *Proc. WiCom*, Shanghai, China, 2007, pp. 21-25.

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

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

[66] C. L. B. Maia, R. A. F. do Carmo, F. G. de Freitas, G. A. L. de Campos, and J. T.

de Souza, "A Multi-Objective Approach for the Regression Test Case Selection Problems," in *Proc. XLI Brazilian Symposium of Operational Research*, 2009, pp. 1824-1835.

[67] P. McMinn, "Search-based Software Test Data Generation: A Survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105-156, 2004.

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

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

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

[71] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "MOCell: A Cellular Genetic Algorithm for Multiobjective Optimization," *Int. J. Intelligent Systems*, vol. 24, no. 7, pp. 726-746, 2009.

[72] J. Neilson, *Usability Engineering*.: Academic Press, 1993.

[73] G. H. L. Pinto and S. R. Vergilio, "A Multi-Objective Genetic Algorithm to Test Data Generation," in *Proc. ICTAI*, Arras, France, 2010, pp. 129-134.

[74] K. Pohl, G. Böckle, and F.J. van der Linden, *Software Product Line Engineering*. New York: Springer-Verlag, 2005.

[75] 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.

[76] K. Praditwong, M. Harman, and X. Yao, "Software Module Clustering as a Multi-Objective Search Problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264-282, March/April 2011.

[77] O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, pp. 203–249, 2010.

[78] O. Räihä, K. Koskimies, and E. Mäkinen, "Generating Software Architecture Spectrum with Multi-Objective Genetic Algorithms," in *Proc. NaBIC*, 2011, pp. 29-36.

[79] D. Rodríguez, M. Ruiz, J. C. Riquelme, and R. Harrison, "Multiobjective Simulation Optimisation in Software Project Management," in *Proc. GECCO*, Dublin, Ireland, 2011, pp. 1883-1890.

[80] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed.: Pearson, 2010.

[81] F. Sarro, F. Ferrucci, and C. Gravino, "Single and Multi Objective Genetic Programming for Software Development Effort Estimation," in *Proc. SAC*, Trento,

Italy, 2012, pp. 1221-1226.

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

[83] A.S. Sayyad, K. Goseva-Popstojanova, T. Menzies, and H. Ammar, "On Parameter Tuning in Search-Based Software Engineering: A Replicated Empirical Study," in *Proc. RESER*, Baltimore, USA, 2013.

[84] 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.

[85] A.S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable Product Line Configuration: A Straw to Break the Camel's Back," in *Proc. ASE*, Palo Alto, USA, 2013, pp. 465-474.

[86] 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.

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

[88] 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.

[89] C. L. Simons and I. C. Parmee, "User-centered, Evolutionary Search in Conceptual Software Design," in *Proc. CEC*, Hong Kong, China, 2008, pp. 869-876.

[90] C. L. Simons, I. C. Parmee, and R. Gwynllyw, "Interactive, evolutionary search in upstream object-oriented class design," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 798-816, Nov/Dec 2010.

[91] 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, USA, 2011, pp. 536-539.

[92] L. Thiele, K. Miettinen, P.J. Korhonen, and J. Molina, "A Preference-Based Evolutionary Algorithm for Multi-Objective Optimization," *Evolutionary Computation*, vol. 17, no. 3, pp. 411–436, 2009.

[93] S.W. Thomas, H. Hemmati, A.E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Software Engineering*, July 2012.

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

[95] V. Veerappa and E. Letier, "Understanding Clusters of Optimal Solutions in Multi-Objective Decision Problems," in *Proc. RE*, Trento, Italy, 2011, pp. 89-98.

[96] R.E. Olaechea Velazco, "Comparison of Exact and Approximate Multi-Objective Optimization for Software Product Lines," University of Waterloo, Waterloo, Canada, Master's Thesis 2013.

[97] H. Wada, P. Champrasert, J. Suzuki, and K. Oba, "Multiobjective Optimization of SLA-Aware Service Composition," in *Proc. IEEE Workshop on Methodologies for Non-functional Properties in Services Computing*, Honolulu, USA, 2008, pp. 368-375.

[98] 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.

[99] Z. Wang, T. Chen, K. Tang, and X. Yao, "A Multi-objective Approach to Redundancy Allocation Problem in Parallel-series Systems," in *Proc. CEC*, Trondheim, Norway, 2009, pp. 582-589.

[100] J. Wang and Y. Hou, "Optimal Web Service Selection based on Multi-Objective Genetic Algorithm," in *Proc. ISCID*, 2008, pp. 553-556.

[101] Z. Wang, K. Tang, and X. Yao, "A Multi-objective Approach to Testing Resource Allocation in Modular Software Systems," in *Proc. CEC*, Hong Kong, China, 2008, pp. 1148-1153.

[102] Z. Wang, K. Tang, and X. Yao, "Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems," *IEEE Transactions on Reliability*, vol. 59, no. 3, pp. 563-575, September 2010.

[103] 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.

[104] 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.

[105] 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.

[106] 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.

[107] J. L. Wilkerson, D. R. Tauritz, and J. M. Bridges, "Multi-objective Coevolutionary Automated Software Correction," in *Proc. GECCO*, Philadelphia, USA, 2012, pp. 1229-1236.

[108] T. Yano, E. Martins, and F. L. de Sousa, "Generating Feasible Test Paths from an Executable Model Using a Multi-Objective Approach," in *Proc. SBST*, Paris, France, 2010, pp. 236-239.

[109] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proc. ISSTA*, 2007, pp. 140-150.

[110] S. Yoo and M. Harman, "Using Hybrid Algorithm For Pareto Effcient Multi-Objective Test Suite Minimisation," *Systems and Software*, vol. 83, no. 4, pp. 689-701, April 2010.

[111] S. Yoo, M. Harman, and S. Ur, "Highly Scalable Multi-Objective Test Suite Minimisation Using Graphics Cards," in *Proc. SSBSE*, Szeged, Hungary, 2011, pp. 219-236.

[112] S. Yoo, R. Nilsson, and M. Harman, "Faster Fault Finding at Google using Multi-Objective Regression Test Optimisation," in *Proc. ESEC/FSE*, Szeged, Hungary, 2011.

[113] Y. Zhang, E. Alba, J. J. Durillo, S. Eldh, and M. Harman, "Today/Future Importance Analysis," in *Proc. GECCO*, Portland, USA, 2010, pp. 1357-1364.

[114] Y. Zhang, M. Harman, and S. A. Mansouri, "The Multi-Objective Next Release Problem," in *Proc. GECCO*, 2007, pp. 1129-1136.

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

[116] 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.

[117] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," in *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*. Athens, Greece, 2001, pp. 95-100.

[118] 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.

Curriculum Vitae

Abdel Salam Sayyad received his B.Sc. in EE from Birzeit University, Palestine in 1998, and his M.Sc. in EE from University of Maryland, USA in 2000. He worked as design engineer at Patton Electronics Company, Maryland, USA from 2000 to 2005, and then as instructor of computer engineering at Birzeit University, Palestine from 2005 to 2011. He joined the Ph.D. program in Computer Engineering at West Virginia University in 2011. He is a student member of IEEE and IEEE Computer Society.