Comparison of Exact and Approximate Multi-Objective Optimization for Software Product Lines

by

Rafael Ernesto Olaechea Velazco

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2013

Abstract

Software product lines (SPLs) manage product variants in a systematical way and allow stakeholders to derive variants by selecting features. Finding a desirable variant is hard, due to the huge configuration space and usually conflicting objectives (*e.g.*, lower cost and higher performance). This scenario can be reduced to a multi-objective optimization problem in SPLs. We address the problem using an exact and an approximate algorithm and compare their accuracy, time consumption, scalability and parameter setting requirements on five case studies with increasing complexity.

Our empirical results show that (1) it is feasible to use exact techniques for small SPL multi-objective optimization problems, and (2) approximate methods can be used for large problems but require substantial effort to find the best parameter settings for acceptable approximation. Finally, we discuss the tradeoff between accuracy and time consumption when using exact and approximate techniques for SPL multi-objective optimization and guide stakeholders to choose one or the other in practice.

Acknowledgements

I thank my supervisor Krzysztof Czarnecki for his help and guidance during my master studies and the writing of this thesis. I also thank Derek Rayside and Jianmei Guo for all their help and guidance during my Master's studies and the writing of this thesis.

Dedication

This is dedicated to my parents Monica and Rafael.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Variability is ubiquitous. Physical products, such as automobiles and mobile phones, are produced as a set of variants, and so is the software embedded in them. *Software Product Line (SPL)* engineering is gaining momentum in academia and industry to effectively manage product variants out of a range of configurable software assets [6]. *Features*, essentially increments of functionality such as password protection for a mobile phone platform, are used to abstract software assets for effective configuration. Features are typically incorporated into a *feature model*, which has a tree-like structure and describes choices that stakeholders can make when configuring an SPL [18, 5]. Stakeholders are also interested in a product's quality attributes, such as weight, cost, and performance. An *attributed feature model* is a feature model extended to describe the contribution of each feature to each quality attribute [4].

Stakeholders select features to derive a desirable *configuration* (*i.e.*, a selection of features) that meets specific functional requirements as well as certain quality attributes. However, finding such a desirable configuration efficiently is a hard task. Since features are functional properties, only after creating a full configuration of such features, are the quality attributes of the configuration known. Moreover, the configuration space of an SPL grows exponentially with the number of features and objectives. Furthermore, when deriving a desirable configuration, we may encounter conflicting objectives, *e.g.*, lower cost and higher performance. Hence, engineers have to make trade-offs between these conflicting objectives.

The above scenarios can be reduced to a *multi-objective optimization* problem with constraints, *i.e.*, minimizing or maximizing a set of quality attributes while providing certain functionalities, given the attributed feature model. We can address multi-objective optimization either *exactly* or *approximately* to find a set of optimal or sub-optimal solutions. Exact approaches have the advantage of accuracy, but often take too long for large-scale problems, whereas approximate methods may solve large-scale problems even in a couple of minutes but suffer from lower accuracy with missed optimal solutions. In order to decide whether to use exact or approximate techniques for SPL multi-objective optimization, it is important to understand the trade-offs between the resources and time required by exact approaches versus the risk of missing relevant and optimal solutions when using approximate techniques.

As a representative of approximate methods, *Multi-Objective Evolutionary Algorithms (MOEAs)* have been recently used to deal with SPL multi-objective optimization [29]. However, to the best of our knowledge, there has been no systematic study on the inherent sensitivity of MOEAs to their parameter settings for SPL multi-objective optimization. Moreover, we are unaware of any other work that applies and evaluates an exact algorithm for SPL multi-objective optimization. Recent advances in Satisfiability Modulo Theory (SMT) solvers present an outstanding performance improvement [7, 27], which encourages us to use an incremental and exact algorithm, called *Guided Improvement Algorithm (GIA)* [26], to investigate its feasibility for SPL multi-objective optimization.

In this thesis, we implement GIA and a popular MOEA, called *Indicator-Based Evolutionary Algorithm (IBEA)* [40], for SPL multi-objective optimization. We systematically compare GIA to IBEA on five case studies of SPL multi-objective optimization, in terms of their accuracy, time consumption, scalability, and parameter tuning requirements.

In summary, we make the following contributions:

- Our empirical results based on five case studies suggest that GIA can produce all exact optimal solutions in less than two hours for small SPL with less than 45 features, while IBEA can produce approximate solutions with an average accuracy of at least 42% in less than 20 minutes even for large SPL with 290 features. Thus, we demonstrate the feasibility of exact algorithms for small-scale SPL multi-objective optimization problems and confirm the advantages of approximate algorithms for large-scale problems.
- We conduct a parameter sweep to systematically analyze the sensitivity of IBEA to its parameter settings, following the guidance from Hadka & Reed [13]. Our empirical results show that IBEA requires substantial effort to find the best parameter setting for acceptable approximation. For our largest case study with 290 features, only 4% out of 1000 parameter settings of IBEA can produce any valid solutions at all.
- Our empirical study helps stakeholders understand the trade-offs between accuracy and time consumption when using exact or approximate techniques for SPL multi-

objective optimization, and guides them to choose one or the other in practice.

1.1 Thesis Organization

Chapter 2 presents background information about attributed feature models and multi-objective optimization. It also introduces a mobile phone product line as an illustrative example, showing the results of applying multi-objective optimization on it.

Chapter 3 describes an exact multi-objective optimization method, the Guided Improvement Algorithm (GIA). We describe our implementation of it using a Satisfiability Modulo Theory solver and how we translate the constraints implied by feature models into propositional logic as required by the Satisfiability Modulo Theory. We illustrate how the GIA would run on the example product line introduced in chapter 2.

Chapter 4 describes an approximate multi-objective optimization algorithm, the Indicator-Based Evolutionary Algorithm (IBEA). It also gives an overview of parameter sampling techniques.

Chapter 5 lists and explains the research questions we investigated. It also describes the five subject models we used, the experimental setup and accuracy metrics to be used in our evaluation. Chapter 6 answers the research questions based on our experiments on the five subject models. It also discuses the different trade-offs between exact and approximate methods for multi-objective optimization of software product lines.

Chapter 7 discusses the threats to validity of our studies and how we mitigated each one of them. It is followed by Chapter 8 which discusses related work both in terms of exact and approximate techniques for multi-objective optimization of software product lines. Finally Chapter 9 highlights conclusions and discusses future work.

Chapter 2

Preliminaries

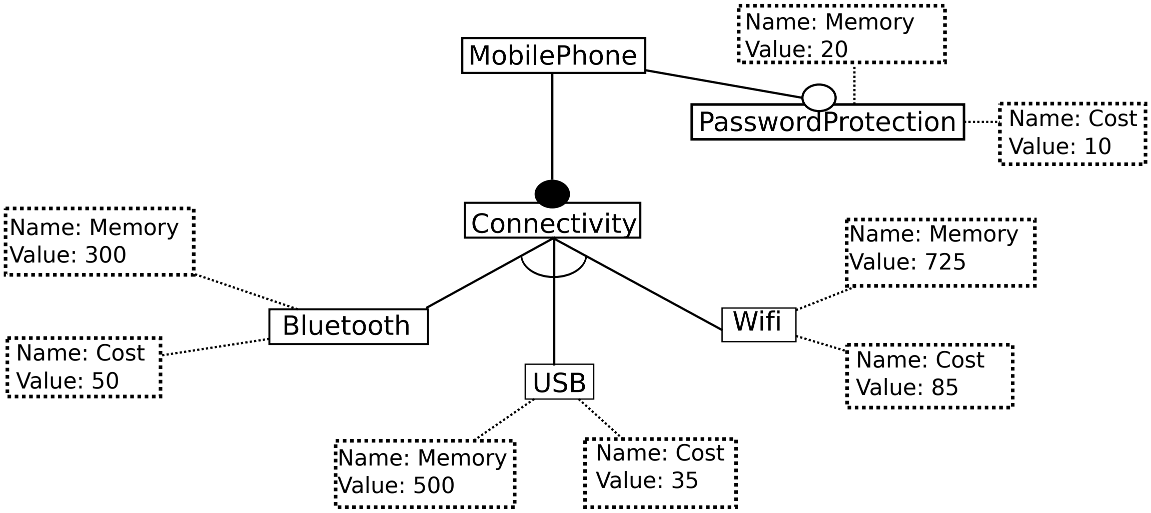
2.1 Attributed Feature Models

Figure 2.1 presents an attributed feature model of a mobile phone platform. The model defines a set of configuration constraints including mandatory (filled circle), optional (empty circle), and alternative (arc). For example, feature Connectivity must be selected in each valid configuration, feature PasswordProtection is optional to be selected, and only one of the three features Bluetooth, USB, and Wifi can be selected.

Each feature is assigned a quality attribute and its value quantifies the impact the feature has on the quality of a product variant that can be measured, such as memory and cost. For example, in Figure 2.1, the impact of feature PasswordProtection on memory consumption is quantified as 20. We quantify and specify quality attributes by simple aggregation functions, such as sum ($e.g.$, $Cost_{Product} = \sum_{feature \in Product} Cost_{feature}$) or multiplication, across the contributions of all features present in a given product as well as relevant feature interactions, which may cause an additional impact on a configuration's quality attributes based on the combined effect of a specific set of features used together [30]. For example if feature PasswordProtection and Wifi were both selected together then the memory consumption could increase by an additional 30 units due to their interaction.

2.2 Multi-Objective Optimization Problem (MOOP)

Multi-objective optimization arises when optimal solutions involve trade-offs between two or more conflicting objectives. For example, Figure 2.2 presents the objective space of the



$$Cost_{Product} = \sum_{feature \in Product} Cost_{feature} \tag{2.1}$$

$$Memory_{Product} = \sum_{feature \in Product} Memory_{feature} \tag{2.2}$$

Figure 2.1: An attributed feature model of a mobile phone platform (adapted from [4])

mobile phone platform shown in Figure 2.1, formed by the two quality attributes, memory consumption and cost.

Stakeholders intend to minimize memory consumption as well as cost to derive an optimal configuration. In general, there is no single solution that simultaneously optimizes each objective, but a set of *Pareto-optimal* solutions, which are optimal in the sense of *Pareto dominance* [32]. A solution dominates another solution when it is better in at least one objective and not worse in all the other objectives. A Pareto-optimal solution is not dominated by any other solution. For example, Figure 2.2 illustrates six valid configurations of the mobile phone platform example in Figure 2.1 and their quality attributes in the objective space. Configuration P1 has the lowest memory consumption and configuration P2 has the lowest cost. They are not dominated by any other solutions and thus they are Pareto-optimal solutions. The set of all Pareto-optimal solutions constitute the *Pareto front* (dotted line in Figure 2.2) of optimal products.

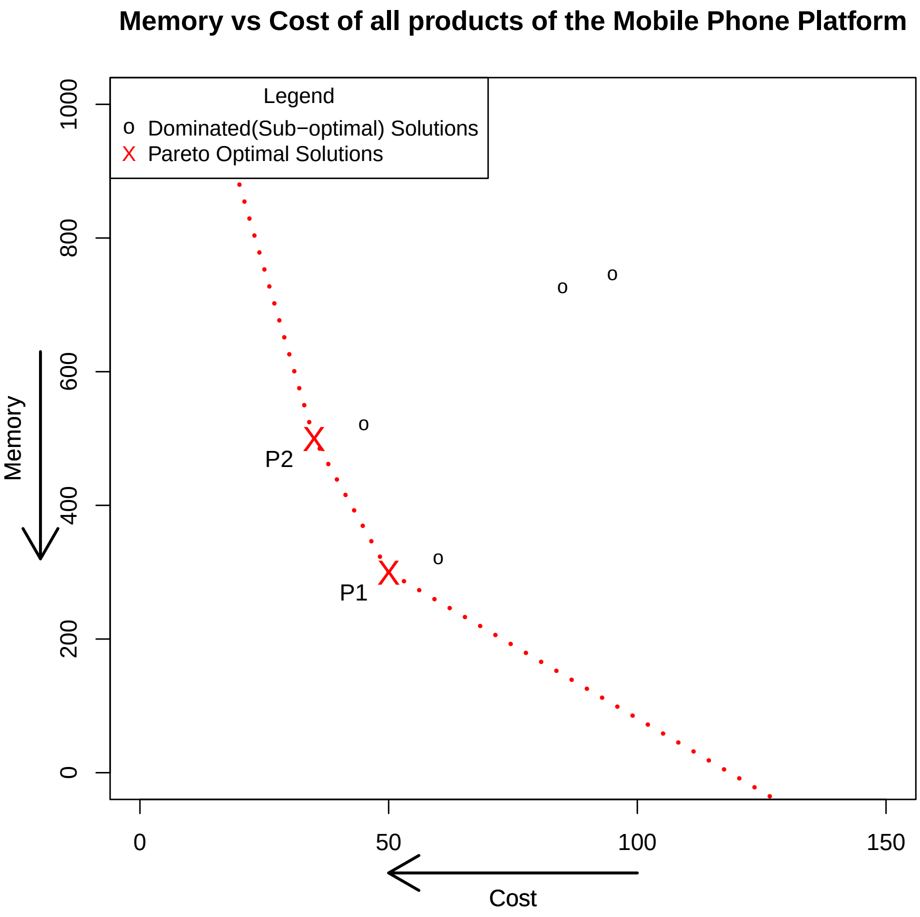


Figure 2.2: The objective space and Pareto front of the mobile phone platform example in Figure 2.1

Chapter 3

Exact MOOP : Guided Improvement Algorithm(GIA)

3.1 GIA Algorithm Description

The Guided Improvement Algorithm (GIA) [26] is an algorithm for exact multi-objective optimization with discrete decision variables. Since features can be abstracted as discrete decision variables, we use GIA for SPL multi-objective optimization.

GIA incrementally explores the objective space and finds the Pareto Front using off-the-shelf solvers. We implement GIA using the Satisfiability Modulo Theory (SMT) solver Z3 [7], due to its outstanding performance in the reasoning and checking of model properties [27]. We first used an implementation of the Guided Improvement Algorithm based on the Alloy solver as a backend, but it was too slow due to the way Alloy represented integers by creating a propositional variable for each possible integer value [17]. Hence we decided to implement GIA using the SMT solver Z3.

GIA works as follows: on each step a candidate solution is replaced by another solution (the candidate solution is excluded from the search by adding a constraint) that dominates it, if one exists. When no more dominating solutions can be found, the current candidate solution is added to the Pareto front and the process is restarted. The pseudo-code for GIA is listed in Algorithm 1 and will be explained in the next section by way of an example exection.

We have used exact solutions computed by our implementation of GIA to evaluate the accuracy of the IBEA heuristic algorithm (introduced in Chapter 4).

Algorithm 1: Guided Improvement Algorithm

```
   input : $Constraint\ F, Metric\ M$
1  $Formula\ notDominated \leftarrow true$
2  $Solution\ s \leftarrow SolveOne(F)$
3  while $s \neq \emptyset$ do
4      while $s \neq \emptyset$ do
5          $s' \leftarrow s$
6          $Formula\ betterMetric \leftarrow buildFormula(\lambda x.dominates(x, s, M))$
7          $s \leftarrow SolveOne(F \wedge betterMetric)$
8      $Formula\ sameMetric \leftarrow buildFormula(\lambda x.equals(x, s', M))$
9      for $a$ in $SolveAll(F \wedge sameMetric)$ do
10         yield $a$
11     $notDominated \leftarrow notDominated \wedge \neg buildFormula(\lambda x.dominates(s', x, M))$
12     $s \leftarrow SolveOne(F \wedge notDominated)$
```

3.2 GIA Example Run

We illustrate a run of the GIA on the mobile phone example from Figure 2.1. In Figure 3.1 we show the different solutions, or products, of the mobile phone platform, labelled as P_1 to P_6 to identify them and be able to refer to each one of them.

At the start of the algorithm any solution can be chosen as the initial solution. For example solution P_6, the mobile phone with exactly features Wifi and Password Protection, could be chosen as the starting solution (line 2). It has a cost of 95 units and memory usage of 745 units. The formula betterMetric would then restrict the search for a solution that dominate P_6, that is it would restrict the search to those solutions that have lower values for cost or memory and not higher on the other one, formally: $(Cost \leq 95 \wedge\ Memory < 745) \vee (Cost < 95 \wedge Memory \leq 745)$.

On the next step (line 7) the solver would look for a better solution, and could find for instance P_4, which has a cost of 45 and memory consumption of 520 units. P_4 dominates P_6 in the sense that it has lower cost and lower memory. The solver would then look for a solution that dominates P_4, that is one with $(Cost \leq 45 \wedge\ Memory < 520) \vee (Cost < 45 \wedge Memory \leq 520)$. The only solution satisfying such constraint is P_2, so the backend solver would return such solution. P_2 has a cost of 35 and memory consumption of 500. The solver would then look for a solution dominating P_2, that is one with :$(Cost \leq 35 \wedge$

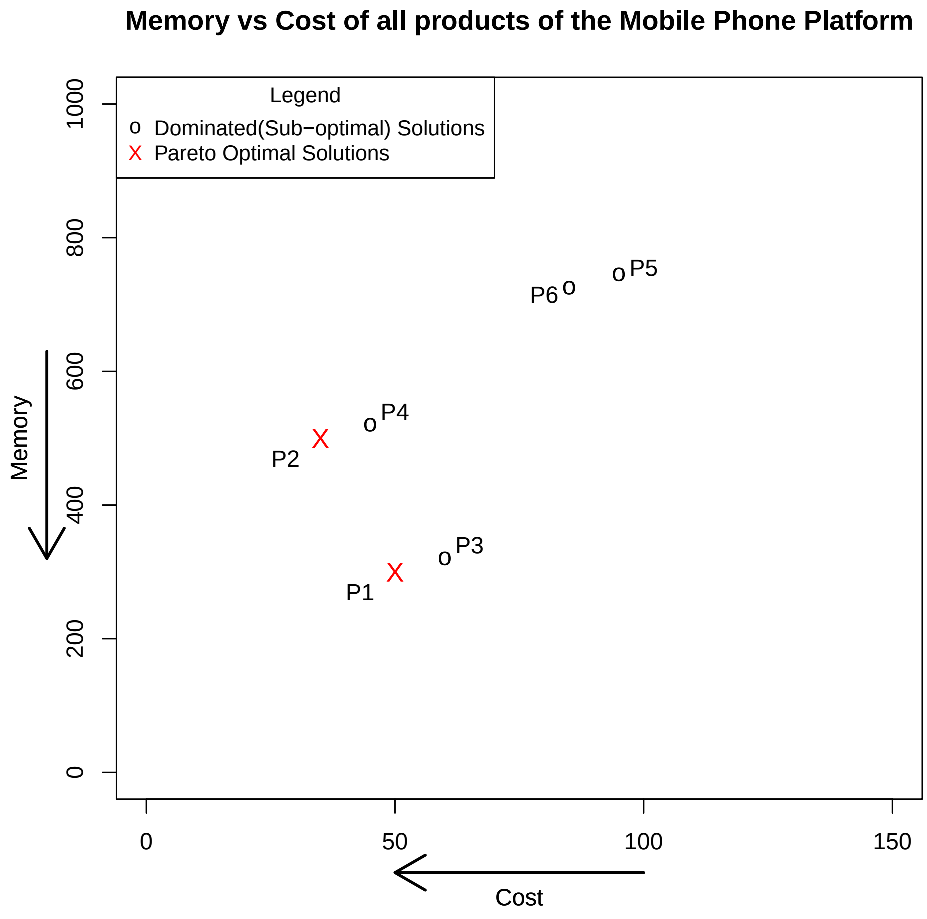


Figure 3.1: Memory vs Cost of all products from the mobile phone platform

$Memory < 500) \vee (Cost < 35 \wedge Memory \leq 500)$. As no such solution exists, the backend solver would return unsatisfiable.

The algorithm now knows that any solution with a cost of 35 and memory consumption of 500 is Pareto optimal, and would add all solutions with such objective values (only P_4 in this case) to the Pareto front (lines 8-10). This step is necessary as there could be multiple products (i.e., multiple combinations of features) that have the same cost and memory values (objective values) and hence are all Pareto-optimal products.

After P_4 has been found to be a Pareto optimal solution, a constraint is added to exclude any solution that is dominated by P_4 from the next iteration (line 11). The constraint will be: $(Cost > 35 \wedge \; Memory \geq 500) \vee (Cost \geq 35 \wedge Memory > 500)$. Then the backend solver would search for a solution satisfying such constraint (line 12). Only P_1 and P_3 satisfy such constraint and the backend solver could return any of those two. For brevity let us assume the solver returns P_1.

P_1 has cost of 50 and memory consumption of 300. The algorithm would look for a solution dominating P_1, that is one with: $Cost \leq 50 \wedge \; Memory < 300) \vee (Cost < 50 \wedge Memory \leq 300)$. As no such solution exists, the backend solver would return unsatisfiable. The algorithm would then look for all solutions with the same objective values as P_1 and add them and P_1 to the Pareto front (lines 8-10). Only P_1 has objective values of 50 and 300, so only P_1 would be added to the Pareto front.

After that, a constraint eliminating all solutions dominated by P_1 from the search space would be added. The constraint will be: $(Cost > 50 \wedge \; Memory \geq 300) \vee (Cost \geq 50 \wedge Memory > 300)$. As no such solution not dominated by P_1 and also not dominated by P_2 exists, no such solution would be found by the solver and it would return unsatisfiable (line 12). Hence the algorithm would terminate as no more Pareto points to be found remain. The complete Pareto front, consisting of solutions P_1 and P_2, has been found incrementally by the GIA.

3.3 Implementation

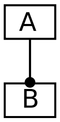
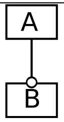
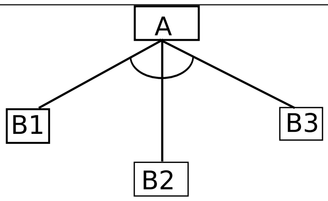
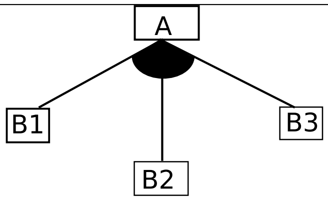
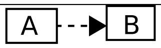
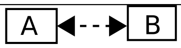
In order to implement the GIA using the SMT solver Z3 as a backend, we expressed each feature as a boolean decision variable. We also translated the constraints from a feature model into propositional logic, following the translation given by Benavides *et al.* [4]. Table 3.1 summarizes such translation. Finally we had to express the objectives in terms of the feature variables, using the arithmetic operators (sum and/or multiply) provided by Z3. We implemented an automatic translator from *attributed feature models* into Z3 specifications.

3.3.1 Attributed Feature Model Translation

We illustrate the translation of an attributed feature model into a Z3 specification, using the mobile phone platform of Figure 2.1 as an example.

First, five boolean decision variables MobilePhone, PasswordProtection, Connectivity, Bluetooth, USB and Wifi are created to express the presence, or not, of each feature.

Table 3.1: Translation of a feature model into propositional logic

Relationship	Feature Model Diagram	Propositional Logic Translation
Mandatory Child		$A \iff B$
Optional Child		$B \Rightarrow A$
Exclusive Or		$\forall i \in 1...N\ B_i \iff (A \wedge \bigwedge_{j\neq i, j\in 1...N} \neg B_j)$
Non-Exclusive Or		$A \iff \bigvee_{i=1}^{N} B_i$
Requires		$A \Rightarrow B$
Excludes		$(A \Rightarrow \neg B) \wedge (B \Rightarrow \neg A)$

```
import z3

solver = z3.Solver()

MobilePhone              = z3.Bool(’MobilePhone’)
PasswordProtection       = z3.Bool(’PasswordProtection’)
Connectivity             = z3.Bool(’Connectivity’)
Bluetooth                = z3.Bool(’Bluetooth’)
USB                      = z3.Bool(’USB’)
Wifi                     = z3.Bool(’Wifi’)
```

Then an implication constraint is added for all five parent-child relationships in the mobile phone platform, such that the child feature can be present only if its parent feature is also present.

```
solver.add(z3.Implies(PasswordProtection, MobilePhone))
solver.add(z3.Implies(Connectivity, MobilePhone))

solver.add(z3.Implies(Bluetooth, Connectivity))
solver.add(z3.Implies(USB, Connectivity))
solver.add(z3.Implies(Wifi, Connectivity))
```

The alternative constraint (exclusive or) in the feature model that exactly one of Bluetooth, USB or Wifi must be selected when Connectivity is selected, is converted into a propositional formula and added into the z3 specification. Each child feature from an alternative constraint will be present only in case that none of its siblings are present and its parent feature is present. For example feature Bluetooth would be present only if USB and Wifi are not present (e.g. are set to false) and its parent feature Connectivity is present.

```
solver.add(Bluetooth==And(Not(USB), Not(Wifi), Connectivity))
solver.add(USB==And(Not(Bluetooth), Not(Wifi), Connectivity))
solver.add(Wifi==And(Not(USB), Not(Bluetooth), Connectivity))
```

Then mandatory children constraints are added to the specification: the variable representing selection of the parent feature is set to equal the one for the mandatory child. In the mobile phone platform Connectivity is the only mandatory children (its parent is MobilePhone).

```
solver.add(MobilePhone==Connectivity)
```

Also a constraint is added to ensure the root feature is selected:

```

solver.add(MobilePhone==True)
```

Feature model constraints are now complete. To incorporate attributes a variable is created for each objective. In this mobile phone example two integer variables cost and memory are created. Each one of these variables are then constrained to equal the sum of the contributions of all selected features.

```
total_cost = z3.Int(’total_cost’)
total_memory = z3.Int(’total_memory’)

```

```
solver.add(total_cost == 10*z3.If(PasswordProtection, 1, 0)
            +  50*z3.If(Bluetooth, 1, 0)
            +  35*z3.If(USB, 1, 0)
            +  85*z3.If(Wifi, 1, 0)
            +  0*z3.If(Connectivity, 1, 0)
        )

solver.add(total_memory == 20*z3.If(PasswordProtection, 1, 0)
            +  300*z3.If(Bluetooth, 1, 0)
            +  500*z3.If(USB, 1, 0)
            +  725*z3.If(Wifi, 1, 0)
            +  0*z3.If(Connectivity, 1, 0)
        )
```

The complete z3 specification, of the Mobile Phone Platform of Figure 2.1, is the following:

```
import z3

solver = z3.Solver()

MobilePhone                = z3.Bool(’MobilePhone’)
PasswordProtection         = z3.Bool(’PasswordProtection’)
Connectivity               = z3.Bool(’Connectivity’)
Bluetooth                  = z3.Bool(’Bluetooth’)
USB                        = z3.Bool(’USB’)
Wifi                       = z3.Bool(’Wifi’)

solver.add(z3.Implies(PasswordProtection, MobilePhone))
solver.add(z3.Implies(Connectivity, MobilePhone))

solver.add(z3.Implies(Bluetooth, Connectivity))
solver.add(z3.Implies(USB, Connectivity))
solver.add(z3.Implies(Wifi, Connectivity))

solver.add(Bluetooth==And(Not(USB), Not(Wifi), Connectivity))
solver.add(USB==And(Not(Bluetooth), Not(Wifi), Connectivity))
solver.add(Wifi==And(Not(USB), Not(Bluetooth), Connectivity))
solver.add(MobilePhone==Connectivity)

solver.add(MobilePhone==True)

total_cost = z3.Int(’total_cost’)
total_memory = z3.Int(’total_memory’)

solver.add(total_cost == 10*z3.If(PasswordProtection, 1, 0)
            +  50*z3.If(Bluetooth, 1, 0)
            +  35*z3.If(USB, 1, 0)
            +  85*z3.If(Wifi, 1, 0)
            +  0*z3.If(Connectivity, 1, 0)
        )
solver.add(total_memory == 20*z3.If(PasswordProtection, 1, 0)
            +  300*z3.If(Bluetooth, 1, 0)
            +  500*z3.If(USB, 1, 0)
            +  725*z3.If(Wifi, 1, 0)
            +  0*z3.If(Connectivity, 1, 0)
        )
```

Chapter 4

Approximate MOOP

The most common way to compute approximate solutions to a multi-objective optimization problem is to use an evolutionary algorithm. Multi-Objective Evolutionary Algorithms (MOEAs) use the ideas of natural selection and evolution from nature to perform computation [10, p. 14-35]. An evolutionary algorithm consists of: an encoding for individual candidate solutions, a crossover and mutation mechanism, a survivor selection mechanism, a parent selection mechanism, an initialization mechanism, and termination conditions. Figure 4.1 shows an overview of the different parts of an Evolutionary Algorithm.

An evolutionary algorithm requires an encoding for each individual in the population. We use a string of bits of length equal to the number of features of the product line, to represent each possible configuration of a feature model [16, p. 70-72]. For example, for the mobile phone platform from Figure 2.1, each configuration would be represented by a string of six bits (as it has 6 features), with 1 meaning a feature is present and 0 meaning it is not. Figure 4.2 shows the encoding for a configuration of the mobile phone platform.

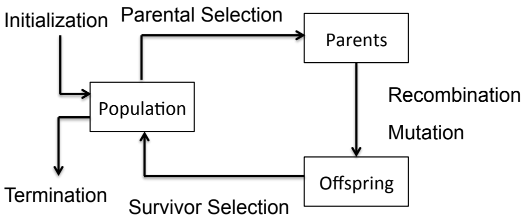


Figure 4.1: A flowchart showing the main steps of an evolutionary algorithm (adapted from [10, p. 17])

MobilePhone	PasswordProtection	Connectivity	Bluetooth	USB	Wifi
1	0	1	0	1	0

Figure 4.2: Encoding a sample configuration for the Mobile Phone from Figure 2.1. This configuration has only features MobilePhone, Connectivity and USB selected.

This bit-string encoding was also used by Sayyad *et al.* [29].

One of the main design decisions when using a MOEA in the SPL domain is how to handle constraints, such as that feature x excludes feature y (or perhaps more complex restrictions). Eiben and Smith [10, p. 205-219] describe three possible approaches:

1. Using a penalty function to de-prioritize solutions that violate constraints, *e.g.*, adding a new metric of the number of constraints violated. Invalid solutions can be removed at termination if desired.

2. Creating a repair operator to ensure every candidate solution is repaired to satisfy constraints.

3. Modifying the combination and mutation operators so that only valid candidate solutions are generated.

Sayyad *et al.* [29] use the first approach in their study of MOEAs for SPLs, and we follow their implementation. We also filter out any non-valid solutions on termination. This approach is easy to implement, but has the potential disadvantage of adding another dimension where the optimization method might get stuck on a local minimum.

MOEAs often have a number of parameters that can be tuned. Hadka & Reed [13] have found that this tuning is important.

The computational budget (b) for a MOEA can be characterized as the cost per generation (c) times the number of generations (g). The number of generations, in turn, can be characterized by the total number of individual fitness evaluations (e) divided by the population size (p). The cost per generation is usually linearly proportional to the size of the population (*i.e.*, $c \propto p$). So, $b \propto c \times g$, which usually equals $p \times \left(\frac{e}{p}\right) = e$.

4.1 Indicator-Based Evolutionary Algorithm (IBEA)

4.1.1 Overview

The Indicator-Based Evolutionary Algorithm (IBEA) [40] was designed for multi-objective optimization by incorporating the *hypervolume* concept into its survivor selection mechanism. The hypervolume concept measures the size of the objective space that is dominated by a set of solutions (an approximate pareto front). Independent studies by Hadka & Reed [13] and by Sayyad *et al.* [29] have confirmed that it is one of the best evolutionary algorithms for multi-objective optimization, including multi-objective optimization for SPLs.

4.1.2 Description

The hypervolume concept measures how much of of the objective space is dominated by a set of solutions compromising an approximate pareto front. As the size of the objective space dominated can be infinite, an auxiliary reference point W which denotes the worst possible values each objective can take is used to truncate it. The hypervolume of a set of solutions Q is calculated as follows [8, p. 318]. For each solution $q_i \in Q$, an hypercube v_i is constructed with the reference point W and the solution q_i as the diagonal corners of the hypercube. The reference point W is the worst possible point that the objectives can take. The overall hypervolume V is the volume of the union of all such hypercubes v_i.

IBEA requires a *binary quality indicator* that can compare the accuracy of two approximations of a Pareto front. Zitzler [40] introduces the I_{HD}-indicator based on the concept of hypervolume. It compares two different approximations, A and B, of a Pareto front in terms of the hypervolume of the objective space dominated by one but not by the other one. Hence $I_{HD}(A, B)$ would be equal to the hypervolume of the objective space dominated by B but not dominated by A. Formally:

$$I_{HD}(A,B) = \begin{cases} Hypervolume(B) - Hypervolume(A) & \text{if } \forall b \in B \; \exists a \in A \;\; a \succ b \\ Hypervolume(A+B) - Hypervolume(A) & \text{else} \end{cases} \tag{4.1}$$

In Figure 4.3 we illustrate the I_{HD}-indicator with approximations A and B each consisting of a singleton set of an individual solution.

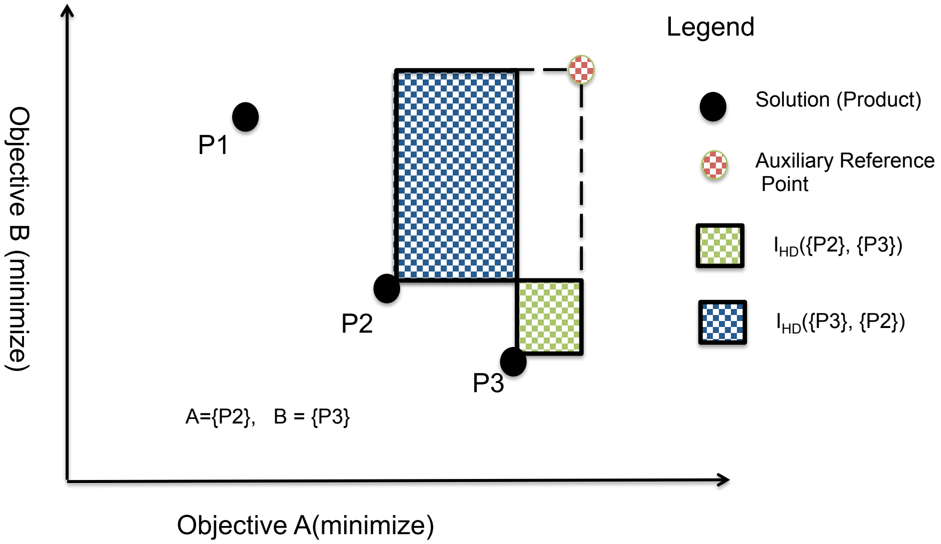


Figure 4.3: Illustration of the I_{HD}-Indicator used for comparing two approximations of the Pareto front each consisting of a single Pareto point.

IBEA assigns fitness values to each individual in the population by aggregating pairwise comparisons of it against each of the other individuals in the population in terms of the I_{HD}-indicator. The formula to compute the fitness of an individual a is:

$$F(a) = \sum_{b \in P \setminus \{a\}} -e^{-I_{HD}(\{a\},\{b\})/\kappa} \tag{4.2}$$

The full IBEA algorithm is listed as Algorithm 2, adapted from Zitzler [40]. IBEA starts by randomly initializing the population P to a set of individuals. It then computes the fitness values of each individual using the I_{HD}-indicator. In case the population size is larger than its maximum allowed size ($P > \alpha$) it filters out the individual with the lowest fitness and updates the fitness values until the population is back to its maximum size. Parents are then selected using standard binary tournament selection. Crossover and mutation operators are applied to the selected parents to generate a new set of individuals as offspring. We fix the mutation operator to be bit-flip mutation and the crossover operator to be single-point crossover. After that the algorithm checks if the maximum number of evaluations have been reached, and if not it iterates again (goes back to line 2 in Algorithm 2).

Algorithm 2: Indicator-Based Evolutionary Algorithm, adapted from [40]

```
input  : $\alpha$  (Population size)
         $N$  (Maximum number of evaluations)
         $\kappa$  (Fitness scaling factor)
output: $A$  (Pareto front approximation)
1 Initialization: Generate an initial Population P; Set the evaluations counter $m$ to 0.
2 Fitness Assignment: Calculate the fitness of individuals in P, for all $x^1 \in P$ set $F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-I_{HD}(\{x^1\},\{x^2\})/\kappa}$ . Increment the evaluations counter $m$ by the size of $P$.
3 Survivor Selection: Iterate the following three steps until the size of population P does not exceed $\alpha$:
4         1. Choose an individual $x^* \in P$ with the smallest fitness value.
5         2. Remove $x^*$ from the population $P$.
6         3. Update the fitness value for the remaining individuals, i.e. $F(x) = F(x) + e^{-I_{HD}(\{x^*\},\{x\})/\kappa}$ for all $x \in P$ .
7 Termination: If $m \geq N$ or some other termination criteria, then set A to the set of non-dominated individuals in P. Stop.
8 Parental Selection: Perform binary tournament selection with replacement on P in order to fill the temporary mating pool $P^{'}$
9 Recombination and Mutation: Apply recombination and mutation operators to the mating pool $P^{'}$ and add the resulting offspring to $P$. Go to step 2.
```

4.1.3 Computational Cost

IBEA differs from many MOEAs in that the cost per generation (c) is quadratic, rather than linear, in the size of the population: *i.e.*, $c \propto p^2$, rather than the usual $c \propto p$. This increase in computational cost per generation is due to computing the hypervolume indicators [35, p. 752]. Therefore, the computational budget of IBEA is characterized by:

$$b \propto c \times g \propto p^2 \times \left(\frac{e}{p}\right) = p \times e$$

4.2 Parameter Sweep: Sobol Sampling

Most MOEAs have a number of adjustable parameters, such as crossover rate, mutation rate, selection strategy, population size, *etc.* Hadka & Reed [13] have demonstrated that most MOEAs, including IBEA, are sensitive to these parameter values: that is, the accuracy of the computed results can vary quite widely depending on the parameters. Currently the determination of which parameter settings are likely to produce accurate results in a given domain is determined empirically by performing a *parameter sweep*: a systematic sampling of the parameter space. *Parameter tuning* techniques aim to find good parameter values for a specific problem instance.

The most commonly-used parameter sweep techniques include Latin Hypercube sampling [23], Stratified sampling [28, p80–82], and Sobol sampling [28, p82–89]. Figure 4.4 illustrates Latin Hypercube sampling and Sobol sampling on a two-dimensional parameter space with 100 sample points. As can be seen visually in Figure 4.4, the Sobol technique samples the parameter space in a more evenly distributed manner.

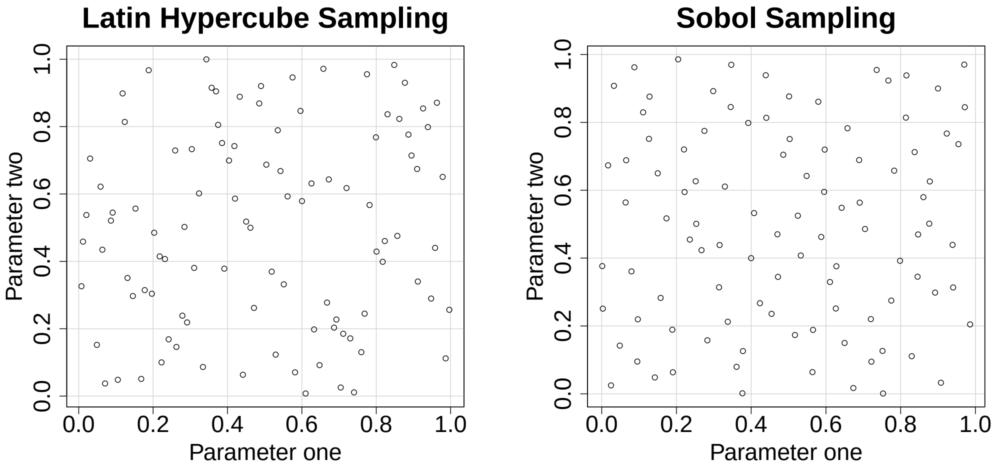


Figure 4.4: Illustration of Latin Hypercube sampling (left) and Sobol sampling (right) in a 2D space

Both Latin Hypercube sampling and Stratified sampling divide the parameter space into segments and then randomly sample within each segment. The intention of this division is to produce relatively even density of samples. Sobol sampling achieves this goal more directly by explicitly minimizing the density differences across the samples. Sobol sampling is the preferred parameter sweep technique for sensitivity analysis [28], and is what we use in this study.

Chapter 5

Experimental Design

5.1 Research Questions

We performed experiments on a collection of SPL attributed feature models to evaluate GIA and IBEA, with a focus on the following research questions:

RQ 1 : Is IBEA more accurate than simple random search?

RQ 2 : How sensitive is IBEA to its parameter settings?

RQ 3 : How fast is IBEA for acceptable accuracy?

RQ 4 : What are good parameter ranges for IBEA?

RQ 5 : How scalable is GIA?

RQ 6 : When is it preferable to use GIA or IBEA?

RQ 1 serves as a baseline to show IBEA is actually doing intelligent search. Sayyad *et al.* [29] showed IBEA was more accurate than other state of the art MOEAs but they did not compare it against random search. Its use is recommended as a minimum by Arcuri & Briand [2] when doing research with randomized algorithms in software engineering: "a search algorithm should always be compared against at least a random search in order to check that success is not due to the search problem (or case study) being easy". Simple random search generates a configuration by selecting each feature with equal probability from a feature model. It discards the configurations that violate any constraints imposed by the feature models and keeps only the valid ones. Then, we calculate the hypervolume of the generated valid configurations to evaluate the accuracy of multi-objective optimization by simple random search.

RQ 2 is designed to investigate our hypothesis that IBEA is highly sensitive to its parameter settings. Harman [15] claims "..., one observation that almost all those who experiment will find, is that the results obtained are often robust to the choice of these parameters. That is, while it is true that a great deal of progress and improvement can be made through tuning, one may well find that all reasonable parameter choices comfortably outperform a purely random search. Therefore, if one is the first to use a search based approach, almost any reasonable (non extreme) choice of parameters may well support progress from the current state of the art". Howevers, we were not convinced this was the case when applying search-based software engineering for multi-objective optimization, based on the findings by Hadka & Reed [13] that showed MOEAs are highly sensitive to parameter settings. Hence we decided to investigate the sensitivity of IBEA to its parameter settings for multi-objective optimization of software product lines.

RQ3 studies the time required to obtain acceptable approximations of the exact Pareto front, taking into account both the time required for tuning the parameters of IBEA and the time IBEA takes to run with these tuned parameters.

The purpose of RQ4 is to detect which parameter ranges are good for IBEA across different software product lines. We study mutation rate, crossover rate and population size.

RQ5 evaluates how well GIA scales in terms of time required versus increasing number of features and objectives.

Finally RQ6 uses the answers to RQ1-RQ5 to synthesize a recommendations as to when should GIA or IBEA be used for multi-objective optimization of software product lines, depending on the number of features and objectives of the software product line.

5.2 Subject Models

We collected a set of attributed feature models from the recent SPL literature (Table 5.1), plus one from another domain (Apollo). These models cover a range of sizes, from a small one such as Berkeley DB, to large ones with hundreds of features such as Eshop, or many objectives, such as ERS.

	#Features	*#CTC*	*#Objectives*
Berkeley DB [30]	12	0	4
Apollo [31]	15	3	2
ERS [11]	35	2	**7**
Web Portal [29, 24]	43	6	4
Eshop [29, 21]	**290**	**19**	4

Table 5.1: List of subject models. 'CTC'=Cross-Tree Constraints. Largest numbers are emphasized.

Berkeley DB [30] describes the price, reliability, security and footprint of a database system. The attribute footprint was measured; the values for reliability and security were inferred (*e.g.*, feature diagnostic would increase reliability); and the values for price were invented. It also contains feature interactions with respect to quality attribute price. For example whenever feature statistics and replication are selected in the same product, then price is increased by an additional 40 units on top of each feature's individual contribution to the price attribute.

Apollo [31] is a model from the engineering design literature that describes the design decisions in the Apollo lunar mission. The quality attributes are cost and mass. This model is technically interesting because the functions used to compute the quality attributes are relatively complex (in the other models the functions are simple summations or in one case multiplications), but can still be encoded into the logics supported by the backend solver Z3. This Apollo model is used with GIA only.

ERS [11] is an Emergency Response System presented with seven quality attributes and objectives: battery usage, response time, reliability, ramp up time, cost, deployment time and development time. The model was built based on expert judgment and used to explore uncertainty in the early architectural design. It included lower, upper bounds and middle values for the contribution of each feature to the quality attributes. We used only the middle values. The reliability attribute is computed by multiplying the reliability of the selected features.

Web Portal [24] and Eshop [21] are feature models describing a product line for web portals and for e-commerce web sites respectively. Sayyad *et al.* [29] augmented these models with three synthetic attributes: cost, priorUsageCount, and defects. Sayyad *et al.* also added an objective to maximize the number of features used. We replicated Sayyad *et al.*'s version of these models, based on their description of how these attributes were randomly generated. Cost was generated as a floating point value between 5.0 and 15.0. It was generated from a truncated normal distribution, with a mean value of 10.0 and standard deviation of 5.0, and lower and upper bounds at 5.0 and 15.0. UsedBefore was a boolean value distributed uniformly between true and false. Defects takes integer values between 0 and 10, and is generated by rounding the values from a truncated normal distribution with mean value of 5.0, standard deviation of 5.0 and upper and lower bounds of 0 and 10. Feature with UsedBefore=False had Defects set to zero as described in Sayyad *et al.* [29], based on the intuition that no defects are yet known for features that have not been used. PriorUsageCount was equal to the number of features that had UsedBefore attribute set to true. So that other researchers can use the same exact values for the synthetic attributes we generated, we list such values in Appendix A.

5.3 Experimental Setup

Both IBEA and GIA have some element of randomness in them, and so multiple runs must be used to get good measurements. This is obvious for IBEA, since randomness is one of the distinguishing features of genetic algorithms. For GIA it is less obvious: each step of the GIA relies on a SAT/SMT solver, which by convention are started with a random seed. Consequently, we ran each algorithm on each subject model multiple times.

We ran the GIA on each of the three smallest models 1000 times (Apollo, Berkeley DB, and ERS). Web Portal was significantly larger, so we ran it only 16 times, which took 26 hours of computation. Eshop was too large to complete a single run, even after several weeks of computation. As a reference set for Eshop we merged the results from all 25,000 runs of IBEA on it, plus also 25 runs of IBEA with the best parameter settings with 2.5M evaluations. In order to merge the results of the different runs for Eshop, we computed the set of non-dominated solutions across the union of all valid solutions produced by each run.

For IBEA, we generated 1000 different parameters settings using Sobol sampling of four parameters: crossover rate from 0 to 1, mutation rate from 0 to 1, maximum number of evaluations considered from 10,000 to 250,000, and population size from 10 to 1000. For each parameter setting, we ran IBEA 25 times with different random seeds on each run.

We also executed random search on each of the problems with varying numbers of instance evaluations between 50M and 550M, increasing the step by 50M each time. We repeated each run 25 times with different random seeds.

As discussed above (§4.1), a good rough measure of the running time budget of IBEA is the population size times the maximum number of individual fitness evaluations.

We ran the GIA on a server with a six-core AMD Opteron 2.8 GHz processor and 32 GB of RAM, for which we had exclusive access. As we had to run the IBEA algorithm for a total of 25,000 different runs, we ran it on a shared cluster (http://sharcnet.ca) of 96 machines each with a quad-core AMD Opteron 2.4 GHz processor and 32 GB of RAM. We did not have exclusive access to this cluster, but for each run IBEA was assigned 1 core and 4 GB of RAM. We used the same cluster for random search.

The implementation of the IBEA algorithm we used was from the JMetal framework [9] (the same implementation used by Sayyad *et al.* [29]).

5.4 Accuracy Metrics

MOEAs, such as IBEA, compute approximations of the Pareto front. Since a Pareto front is, by definition, multi-dimensional, there are a variety of metrics available to measure Pareto front approximations [38, 8, p. 306-324]

Each metric characterizes the approximation's accuracy differently. We use the following two metrics: *Hypervolume Ratio* and *Coverage*.

The *two sets Coverage* metric, used in conjunction with the exact Pareto front, is the number of exact Pareto points included in the approximation [39, 20]. Given an exact Pareto front P and an approximate Pareto front A, then the Coverage of A to P is:

$$Coverage_P(A) = \frac{|\{p \in P \land a \in A : a = p\}|}{|P|} \tag{5.1}$$

Suppose the exact Pareto front has 10 points, and that the approximation contains 4 of these points: then we would say that the Coverage is 40%. Suppose, alternatively, that the approximation contains none of the exact Pareto solutions but contains 10 solutions that are very close to the exact solutions: the Coverage of this approximation would be 0%, but it might score highly on the other metrics.

The *Hypervolume Ratio* [33, 39] is the ratio of the hypervolumes of the approximate Pareto front and the exact Pareto front. An approximation that scored 0% on the coverage metric but was actually quite close to the exact Pareto front would score highly on the hypervolume ratio.

Figure 5.1 illustrates the hypervolume ratio in a two dimensional space. The hypervolume of the true Pareto front is shaded, whereas the hypervolume of the approximate Pareto front is cross-hatched. The hypervolume ratio metric is the ratio of these.

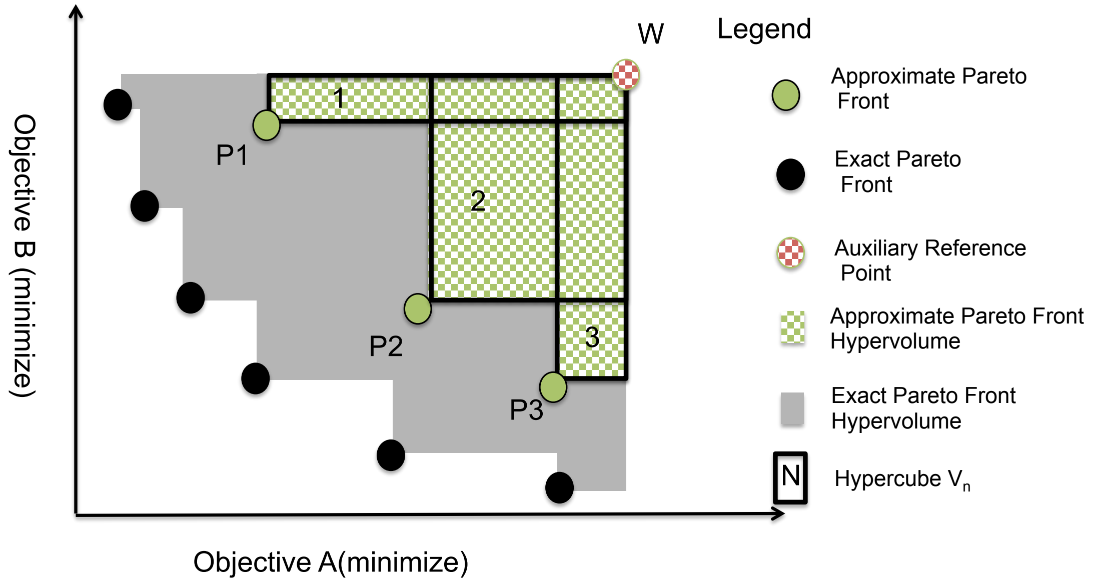


Figure 5.1: The hypervolume for an approximate Pareto front. We show the hypervolume for an approximate Pareto front that consists of the non-dominated solutions P1, P2 and P3. It is the union of the 2-dimensional hypercubes (*e.g.*, rectangles) V_1, V_2, and V_3, that are formed between P_i and the reference point W. The reference point W represents the worst possible value for each objective.

Chapter 6

Empirical Results and Discussion

In this chapter we present the results of our experiments in reference to the six research questions described in chapter 5.

6.1 Is IBEA more accurate than random search?

We compare the hypervolumes obtained by random search versus those obtained by some of 1000 different parameterizations of IBEA. We compare the 100%, 75% and 50% best parameterization of IBEA against random search with 550M number of evaluations. By the X% best parameter setting we mean: the one that ranked in the X percentile across all parameter settings in terms of average hyper volume ratio. Table 6.1 shows descriptive statistics (mean and standard deviations) of the accuracy, measured by the hypervolume ratio metric, and the time taken by IBEA and random search.

There is no reason to believe that the hypervolume produce by either IBEA or simple random search is distributed normally. To mitigate this issue, we perform statistical hypothesis testing using the *two-tailed Mann-Whitney U-test* [37] and the *Vargha-Delaney indicator* [34], according to the guidelines from Arcuri & Briand [2]. We used the non-parametric Mann-Whitney U test to detect statistical differences between the accuracy of IBEA and random search, and the Vargha-Delaney indicator $A_{1,2}$ [34] to report standardized effect sizes in such comparison. Table 6.2 shows the $A_{1,2}$ indicator for such comparison.

We use the two-tailed *Mann-Whitney U-test* to detect the statistical differences between IBEA and simple random search on the hypervolume-based accuracy. This test measures whether there are differences in the stochastic ranking of the values of a metric (*i.e.*,

	Hypervolume (%) (Mean ± Std. Dev.)				Time (Mean ± Std. Dev.)			
	$IBEA_{100}$	$IBEA_{75}$	$IBEA_{50}$	*Random*	$IBEA_{100}$	$IBEA_{75}$	$IBEA_{50}$	*Random*
Berkeley DB	100±0	81±17	61±0	100±0	19.7m±2.1	4.0m±0.0	1.0m±0.1	1.0s±0.0
ERS	42±17	0.2±1	0±0	31±9	1.1h±0.1	0.7h±0.1	0.3h±0.0	7.7h±0.1
Web Portal	92±1	34±14	4±9	72±2	18.5m±0.3	5.8m±0.1	27.9m±1.9	14.6h±0.1
Eshop	78±5	0±0	0±0	0±0	16.7m±0.5	4.3m±0.2	7.4m±0.2	6.5d±0.1

Table 6.1: Hypervolume and time consumption of IBEA using its 100%, 75%, and 50% best parameter settings and of simple random search (s—seconds, m—minutes, h—hours, d—days)

	$A_{100,RS}$	$A_{75,RS}$	$A_{50,RS}$
Berkeley DB	0.5	0.1*	0*
ERS	0.8*	0*	0*
Web Portal	1.0*	0*	0*
Eshop	1.0*	0.5	0.5

Table 6.2: Statistical analysis of the accuracy of IBEA with different parameter-settings versus Random Search, using the Mann-Whitney U test. $A_{IBEA,RS}$ shows the effect sizes and can be interpreted as an estimate of the proportion of runs in which IBEA will give a more accurate answer than random search. All comparisons marked with asterisks showed statistically significant results with $p = 0.000001$.

hypervolume) produced by two algorithms (*i.e.*, IBEA and simple random search). Here, the null hypothesis is that there are no differences, and the alternative hypothesis is that such differences exist, *e.g.*, IBEA tends to produce higher-ranked values of the metric than simple random search.

Moreover, we use the Vargha-Delaney indicator to report *standardized effect sizes* of the above testing, which measure stochastic superiority of one algorithm against the other, that is, the probability of producing a better answer (*e.g.*, higher hypervolume) using one algorithm (*e.g.*, IBEA) instead of the other (*e.g.*, simple random search).

The values of $A_{IBEA-100,RS}$ in Table 6.2 show that IBEA, with its best parameters, is much better than random search. According to $A_{IBEA-100\%,RS}$ IBEA-100% would obtain more accurate answers than random search 100% of the time for Eshop and Web Portal, and 80 % of the time for ERS. The better accuracy of IBEA-100% compared to random search is also supported by the average hypervolumes obtained by IBEA-100% and random search where IBEA obtains 78%, 92% and 42% of hypervolume versus 0%, 72% and 31% for random search for the product lines Eshop, Web Portal and ERS, respectively. This demonstrates that IBEA, with its best parameters, is much more accurate than random search. It is also several orders of magnitude faster than random search as can be seen in Table 6.1.

However, not all parameterizations of IBEA are more accurate than random search. The values of $A_{IBEA-75,RS}$ indicate that random search is much better than IBEA-75% as it will obtain more accurate answers than it 100% of the time for Web Portal and ERS, and 90% of the time for Berkeley DB. Moreover the average values of the hypervolume ratio obtained by random search also point to its superiority in terms of accuracy over IBEA-75%. This is also supported by Figure 6.2(a) where we observe that the accuracy of IBEA is highly sensitive to the parameter settings used.

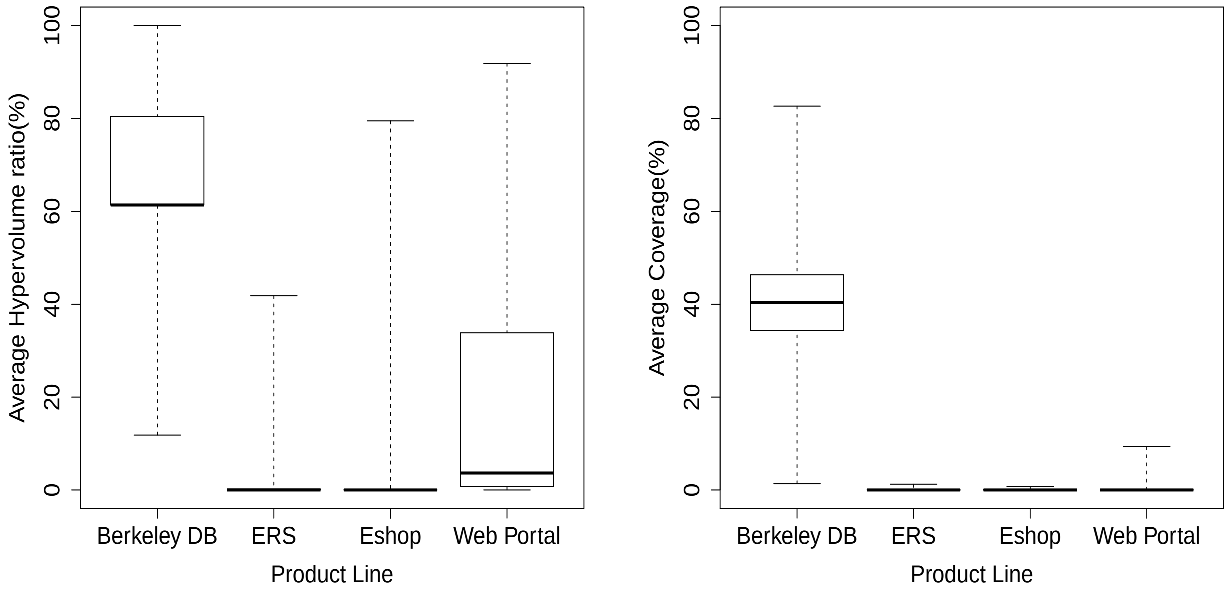
In conclusion, IBEA can be both more and less accurate than random search depending on the parameters used. IBEA must be used with some kind of parameter tuning technique.

6.2 How sensitive is IBEA to its parameters?

Figures 6.1(a) and 6.1(b) show box-plots of the average hypervolume ratio and average coverage across 1000 different parameter settings for each subject model. For both ERS and Eshop, more than 75% of all parameter settings obtain a hypervolume ratio and coverage of zero, and more than 75% of parameter settings for Web Portal produce a coverage of zero.

Furthermore, we ranked the 1000 parameter settings of IBEA from the worst case to the best in terms of their obtained average hypervolume and average coverage. We show such ranking in Figure 6.2. For all subject models except the smallest case, Berkeley DB, IBEA produces acceptable approximations of the Pareto front only if it happens to choose a parameter setting in a small portion of the parameter space (20%, 5% and 2% respectively) in terms of hypervolume, and an even smaller portion of the parameter space (less than 15%, 1% and 1% respectively) in terms of coverage.

From Figures 6.1 and 6.2, we can observe that the accuracy of IBEA is highly sensitive to its parameter settings, which is in line with the findings from [3].



(a)

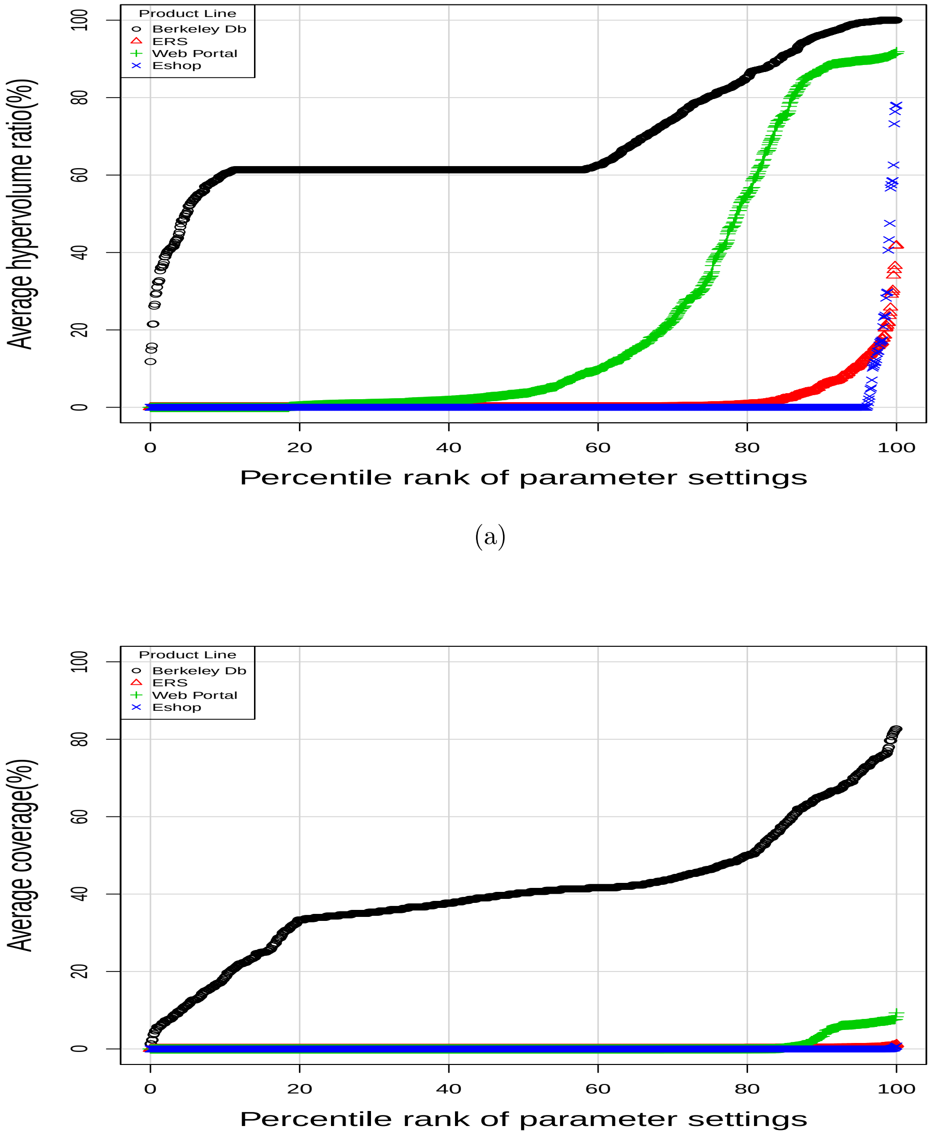
(b)

Figure 6.1: Average hypervolume ratio and coverage across 1000 different parameter settings. Whiskers denote the best and worst parameters.

6.3 How fast is IBEA for acceptable accuracy?

With the best parameter settings, IBEA can produce acceptable approximations of the Pareto front (> 80% hypervolume) for models with 4 or fewer objectives (Figure 6.1(a)). These solutions might even include a number of points from the exact Pareto front (Figure 6.1(b)). But with poor parameter settings IBEA can also produce worthless solutions (Figure 6.1). How long does it take to find and execute good parameter settings for IBEA? This depends on the parameter tuning technique.

The worst case is to do parameter tuning by Sobol sampling (which is intended for sensitivity analysis). How many Sobol samples do we need to take before we can expect to find parameters that produce an accurate result? Figure 6.3 shows the best hypervolume ratio obtained for a given number of Sobol samples, from 0 to 1000. We see that for Berkeley DB, the smallest and simplest model, 100% hypervolume ratio is obtained after only 10 Sobol samples. The next most complicated model, Web Portal, gets close to its best



(a)

(b)

Figure 6.2: Average hypervolume ratio and coverage obtained by IBEA based on percentile ranking of 1000 parameter settings

hypervolume ratio in 50 Sobol samples. The more challenging models, ERS and Eshop, get close to their best in 500 Sobol samples, but keep improving all the way up to 1000 Sobol samples. The time taken for this approach varies from 10 hours for Berkeley DB to 1000 hours for the more complex models (ERS and Eshop). Sobol sampling is an expensive tuning strategy.

A slightly better tuning strategy is to perform Sobol sampling only in parameter ranges that are known to be good for the given problem domain. Our experiments (§6.4) show that mutation rates above 0.2 are rarely good. This cuts down the parameter space by a factor of five, which would reduce the times down to around 2 hours for Berkeley DB and 200 hours for the more complex models.

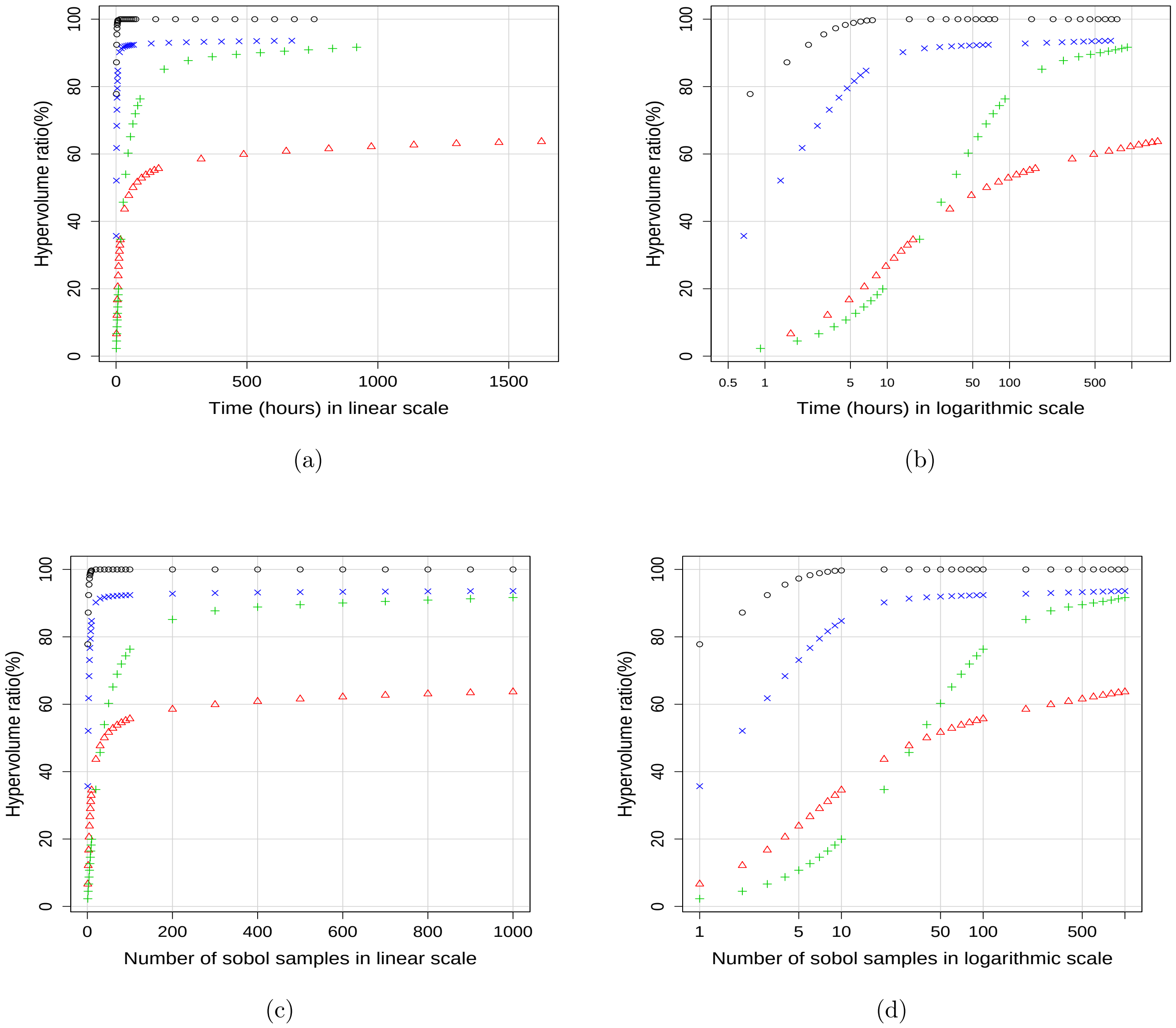
At the other extreme, we can assume a tuning oracle that gives us the best parameter settings. Table 6.1 shows the time taken for single runs on IBEA with different parameter settings: IBEA_{100} is the most accurate parameter setting; IBEA_{75} is the first/second quartile boundary run; and IBEA_{50} is the midpoint run. A tuning oracle would give us the settings for the IBEA_{100} run. The IBEA_{100} runs have times varying from 16.7 minutes to 66 minutes (1.1h). It is interesting to note that these times do not correlate with number of features: Eshop, with the largest number of features (290), has the fastest time. The worst time (66 minutes) belongs to the model with the greatest number of objectives: ERS, 7 objectives.

The IBEA_{100} run is the single most accurate run. Are there other runs with similar accuracy and significantly lower run times? Yes. Figure 6.4 bins the runs according to their budget ($p \times e$) quartile. We see that both Berkeley DB and Web Portal can get very close to their maximum accuracy at their lowest budget ranges (less than 3 minutes). The more complex models, ERS and Eshop, require their highest budget ranges to get their best accuracy.

In summary, IBEA requires several minutes to achieve good accuracy for even the smallest models. The time required appears to be more a function of the number of objectives than the number of features.

6.4 What are good parameter ranges for IBEA?

As described above, we considered four parameters for IBEA: mutation rate, crossover rate, population size (p), and the total number of individual fitness evaluations (e). The strongest conclusion that can be drawn from analyzing our data is that mutation rates over 0.2 almost always lead to poor results — which is consistent with general guidance



(a)

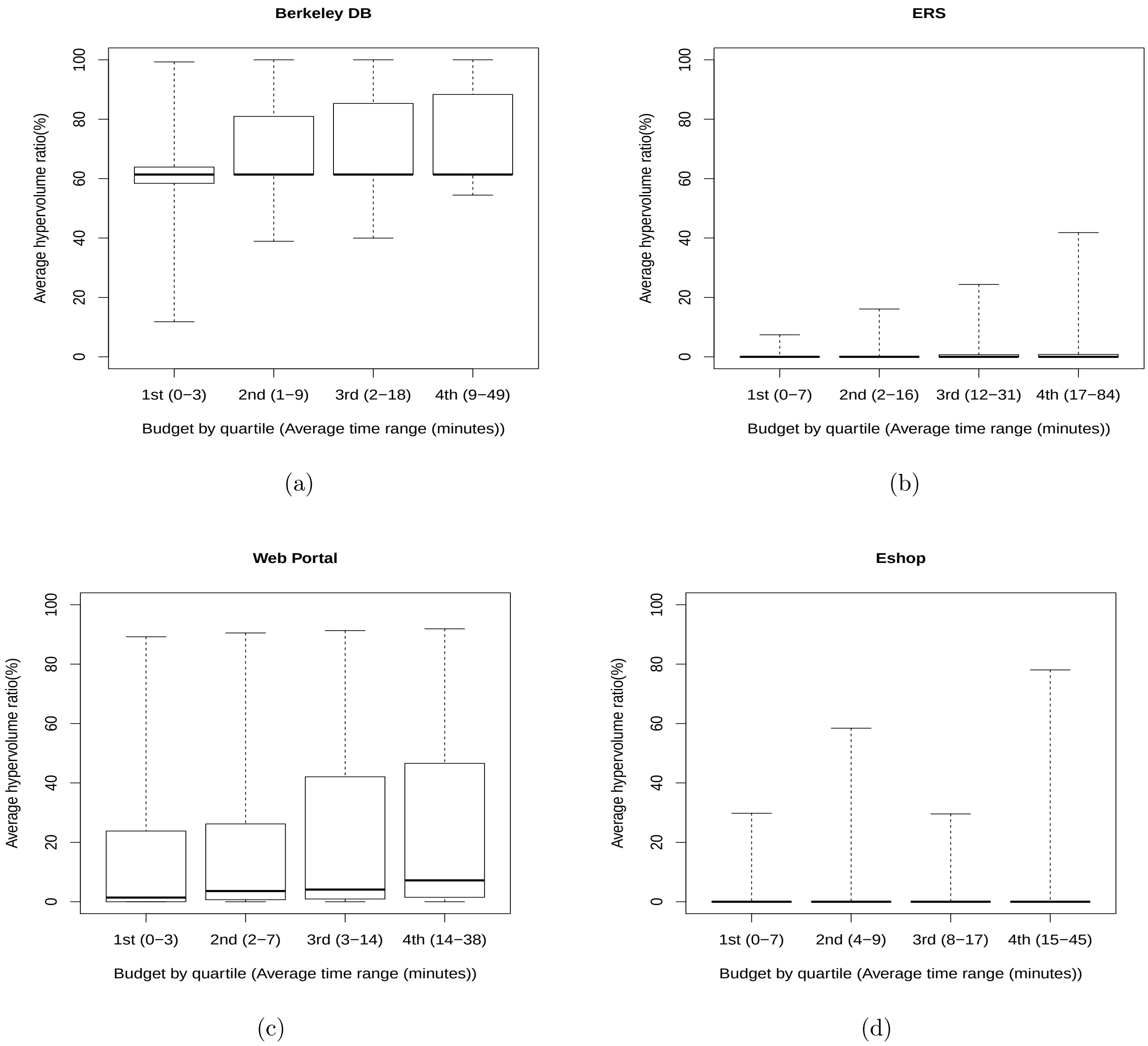
(b)

(c)

(d)

Figure 6.3: Maximum hypervolume ratio (%) and time consumption produced by IBEA using different number of Sobol samples of parameter settings.
Berkeley DB: ○ ERS: △ Web Portal: × Eshop: +

on MOEAs. Figure 6.5 shows a plot of average hypervolume ratio against mutation rate



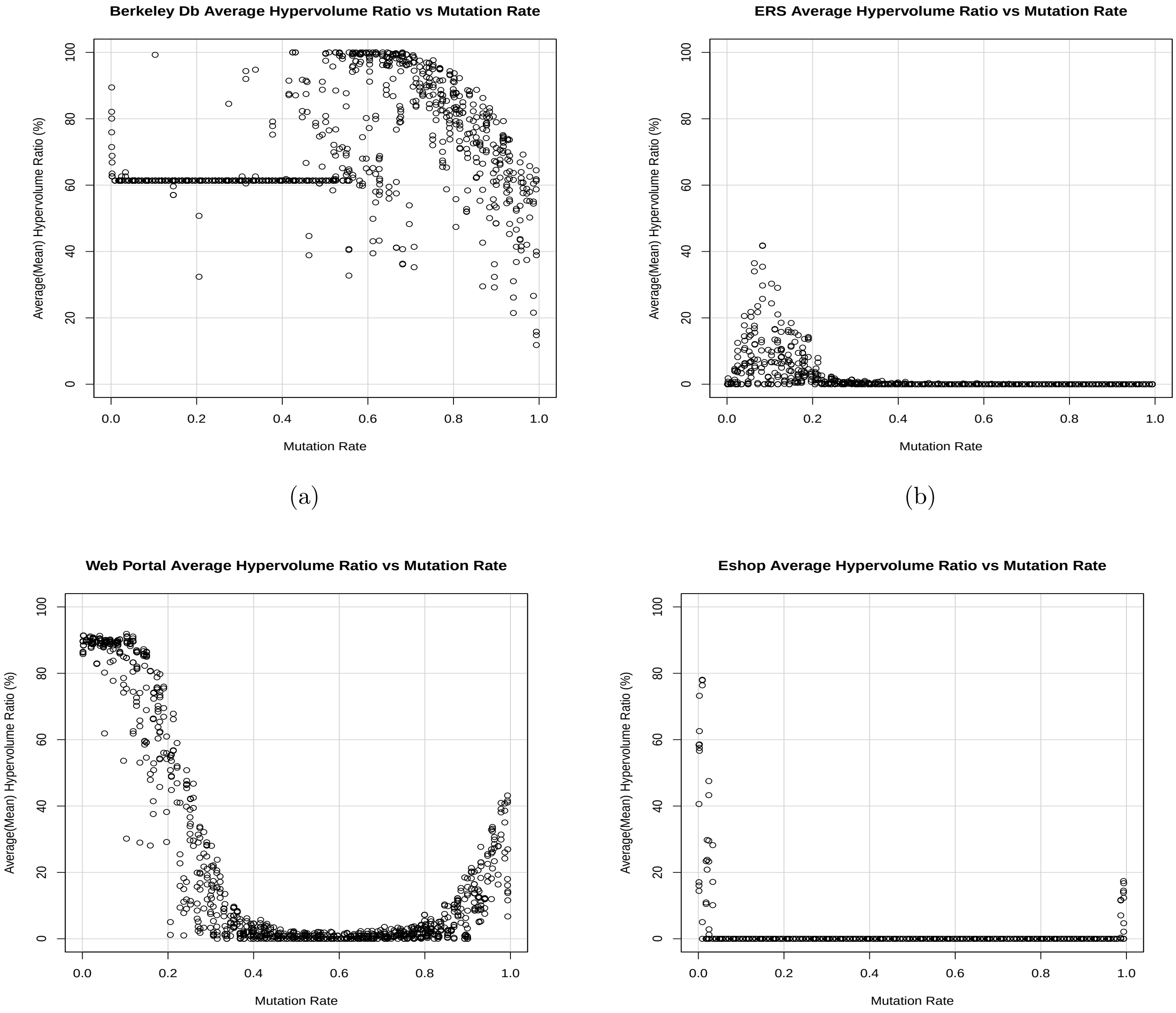
(a) (b)

(c) (d)

Figure 6.4: Average Hypervolume Ratio of IBEA in different ranges of time budget. Whiskers denote the best and the worst cases.

for the four subject models.

We found accurate results at all crossover rates, with no clear trend about what values



(a)

(b)

(c)

(d)

Figure 6.5: Average Hypervolume Ratio Metric versus mutation rates for Berkeley DB, ERS, WebPortal and Eshop.

are better or worse: the entire range of crossover rates (0–1) should be sampled.

For the smaller models we found that larger budgets ($p \times e$, §4.1) produced better

	Pareto Front Size	Time (Mean ± Std. Dev.)
Berkeley DB	12	0.04s ± 6.5%
Apollo	7	1.60s ± 11.7%
ERS	356	32.24s ± 5.2%
Web Portal	890	1.65h ± 6.7%
Eshop	>1	> 15d

Table 6.3: Time consumption of GIA

results (Figure 6.4). This monotonic relationship did not hold for the largest and most constrained model: Eshop. At all budget levels, our implementation of IBEA produced mostly illegal configurations. This result calls into question the decision to handle constraint violations with a penalty function. Given this design decision, which was also used by Sayyad *et al.* [29], then our results show that a variety of budget levels (*i.e.*, values of p and e) should be sampled.

6.5 How scalable is GIA?

Table 6.3 shows the results of running GIA on the five subject models. The second column records the size of the Pareto Front, *i.e.*, the number of Pareto-optimal solutions. The third column collects the mean and standard deviation of the time consumption of running GIA 1000 times for each subject model.

From Table 5.1 and Table 6.3, we can see that GIA can find the Pareto front of an attributed feature model with less than 45 features and up to 7 objectives in a reasonable amount of time (at most 1.65 hours), but it fails for large models with hundreds of features like Eshop.

For Berkeley DB, GIA runs instantly. For the other subject models, GIA may run faster or slower than IBEA, depending on the parameter settings of IBEA.

6.6 When is it preferable to use GIA or IBEA?

GIA is better for models with fewer features (*e.g.*, Berkeley DB, ERS) or higher numbers of objectives (*e.g.*, ERS). We can draw this conclusion by contrasting GIA results in Table 6.3

with IBEA results in Table 6.1, Figure 6.3, and Figure 6.4. GIA actually runs faster than IBEA for the smaller models.

IBEA is clearly better for Eshop, which has the most features in our study: GIA effectively times out on this model (Table 6.3).

The only model for which the result is not immediately obvious is Web portal. GIA computes an exact answer for Web portal in 1.65 hours (Table 6.3). If we had a tuning oracle for IBEA we could get 92% of the hypervolume of Web portal in under 20 minutes (Table 6.1). But we do not have a tuning oracle. The best that we know how to do now is Sobol sampling with known good parameter ranges, and that will give us a run time of around 20 hours for Web portal (§6.3). So, upon consideration, due to the tuning cost of IBEA it is better to use GIA for Web portal.

We can summarize the results for these four models in the following general rule: if GIA completes within a couple of hours then it is the better choice. Otherwise, IBEA is required. GIA is likely to complete within a couple of hours for models with fewer than 45 features, regardless of the number of objectives. A feature of GIA is that it gives Pareto points incrementally. From our experience, if the first Pareto point appears within a few minutes then the entire front will be found within a few hours.

Chapter 7

Threats to Validity

Arcuri & Briand [2] recommend running each configuration 1000 times in order to get a good sample of the distribution of the results. We performed these 1000 runs for the GIA on the smaller models. On the larger WebPortal model we ran the GIA only 16 times, because this already took 26 hours. Similarly, for IBEA we ran each configuration only 25 times due to cost concerns. It is possible that more runs would have produced a more accurate characterization of the accuracy and computing time of the algorithms. The standard deviations of the running times that we did measure are relatively low, so we think there would likely not be much change in our estimate of the running times. The accuracy of IBEA was also fairly consistent in our runs, with the possible exception of ERS, where we observed a high standard deviation. In the hypothetical case that more runs of IBEA for each configuration of ERS would have produced a more accurate result, this would not substantially change the answers to our research questions: IBEA was already much more accurate than random search (§6.1); we would still conclude that IBEA is very sensitive to its parameters (§6.2), and that GIA is a better choice for ERS than IBEA is (§6.6). If more runs for IBEA on ERS produced more accurate results, then we might increase the amount of time that we estimate is required for IBEA to produce an accurate result (§6.3).

We could have produced more detailed answers for research questions 2 (§6.2) and 4 (§6.4) if we had used the variance decomposition method [28] to assess both the first and second-order effects of each parameter on IBEA. This greater level of detail would not have changed our conclusions greatly: IBEA is sensitive to its parameters, and some form of tuning must be done. Hadka & Reed [13] used the variance decomposition method in their study.

Sayyad *et al.* [29] used generated values for the synthetic attributes in the Eshop and

WebPortal product line models. We followed the same technique they describe to generate attribute values, but we couldn't use the exact same attribute values as they were not available. There is some small possibility that the values we generated are somehow importantly different from the values that Sayyad *et al.* [29] generated, in which case it might be difficult to compare some of the measurements in this paper with those in their paper.

Our estimates of how long it takes IBEA to produce accurate results (§6.3) were done without using a clever tuning strategy. The tuning strategy we used was Sobol sampling of known good parameter ranges (§6.4). Perhaps a more clever tuning strategy could produce similar results in less time. We are not aware of a significantly more clever tuning strategy, and have made some modest efforts to look for one (*e.g.*, [22]). A better tuning strategy would not, however, change our conclusion about when to use GIA and when to use IBEA (§6.6): GIA is clearly better for the problems that it can solve, and IBEA is the only choice for the problems that GIA cannot solve.

We used the JMetal [9] implementation of IBEA, which was also used by Sayyad *et al.* [29]. In reading the JMetal source code we noticed some opportunities for potential optimizations, but we do not think that they would have had a significant impact on running time, and even if they did it would not change our conclusion for RQ6 (§6.6).

Finally, the generality of our results is potentially limited by the generality of our subject models. Our study has twice as many subject models as Sayyad *et al.* study [29], and includes all of the models from that study. An important limitation of our subject models is that they employ relatively simple arithmetic to evaluate the metric functions, and these metric functions can be incorporated into the search procedure (which is what the GIA does). This property holds for many of the multi-objective SPL models we are aware of, but it does not hold for SPL models involving certain quality attributes that require simulation (*e.g.*, some models involving performance or reliability attributes) or in other disciplines. For example, in Civil Engineering (*e.g.*, [13]), models often involve metric functions with large differential equations that take hours to solve and cannot be incorporated into the search procedure. Our results do not generalize to those other domains or SPL models. We included the Apollo [31] model from the Engineering Design literature in our study because it employs more sophisticated metric functions than our SPL models, but less sophisticated than many Civil Engineering models. Our main comparative result holds for this model: the GIA is better than IBEA for small models, even with moderately more complicated metric functions.

Our main conclusion about IBEA, that it must be used with a tuning strategy, does not appear to be threatened by the limited generality of our subject models, and is also strongly supported by Hadka & Reed [13].

A broader selection of subject models, in combination with the variance decomposition method [28], might give more precise parameter ranges for using IBEA in SPL. Our study is currently the most precise study of IBEA parameter settings on SPL models that we are aware of.

Chapter 8

Related Work

We discuss related work in terms of both exact and approximate multi-objective optimization for SPLs.

8.1 Exact MOOP for SPLs

Many exact techniques have been applied and extended for optimized feature configuration in SPLs. Benavides *et al.* [4] considered resource constraints in product derivation process and applied Constraint Satisfaction Problems (CSP) techniques to model and solve the optimized configuration problem automatically. They implemented their approach using the Choco constraint solver [1]. Karatas *et al.* [19] further introduced a mapping from attributed feature models to constraint logic programming over finite domains, and thereby facilitating the use of optimization operators provided by constraint logic programming tools. However, existing exact techniques focus on single-objective optimized configuration in SPLs. To the best of our knowledge, our work is the first that evaluates an exact technique for multi-objective SPLs.

8.2 Approximate MOOP for SPLs

SPL configuration optimization has been proven to be an NP-hard problem [36]. To address this problem, exact algorithms often suffer from exponential complexity, but typically there are a number of approximate algorithms with acceptable optimality [12]. Search-Based

Software Engineering [14] advocates the application of optimization techniques from the operations research and heuristic (or metaheuristic) computation research communities to software engineering. It is gaining momentum in academia and industry [14]. Following the same idea, some optimization techniques have been used for SPL optimized configuration.

White *et al.* [36] provided a polynomial time approximation algorithm for selecting a highly optimal set of features that adheres to a set of resource constraints. They proposed Filtered Cartesian Flattening to transform the optimized configuration problem into the multi-dimensional multi-choice knapsack problem, and then use a heuristic technique to produce approximate feature configurations. Guo *et al.* [12] first proposed a genetic algorithm to solve the SPL optimized configuration problem. They introduced a repair operator that can fix a randomly-generated feature configuration to a valid one. Most metaheuristics can be used for SPL optimized configuration. These works examined only the single-objective case.

In previous work [25] we implemented, but did not compare against alternative techniques, GIA for multi-objective SPLs using a SAT solver as the underlying backend, through an intermediate relational solver *kodkod*. We noticed that the encoding of integers used by kodkod to translate integer constraints into propositional logic was creating very large formulas and slowing down our approach. Hence, in this work we improved the performance of GIA by re-implementing it with the Z3 Satisfiability Modulo Theory solver, and evaluated it in comparison with the approximate method IBEA.

Sayyad *et al.* [29] experimented with five MOEAs for SPL multi-objective optimized configuration. They concluded that IBEA is the one that scales better when increasing the number of objectives. However, they did not systematically evaluate IBEA spanning different parameter settings, nor did they compare MOEAs against random search. In contrast, we performed Sobol sampling to evaluate the sensitivity of IBEA comprehensively. Moreover, we compared IBEA to an exact technique and simple random search.

Esfahani *et al.* [11] developed the ERS case study that we used in our experiments. They used a weighed sum of design objectives, instead of a typical multi-objective optimization approach, to search for the optimal design. We converted their case study into an attributed feature model and applied both exact and approximate multi-objective optimization algorithms.

Siegmund *et al.* [30] developed a methodology of quantitatively measuring and specifying quality attributes of SPLs and applied it to several case studies, most of them with either one or at most two quality attributes. We used their Berkeley DB case study.

Chapter 9

Conclusion and Future Work

We have demonstrated that it is feasible to compute exact solutions for multi-objective software product line models with less than 45 features. Previously exact techniques have been applied to only single-objective SPL models (*e.g.*, [4, 19]), and it was assumed that approximate techniques were required for multi-objective models (*e.g.*, Sayyad *et al.* [29]). Our results confirm that larger multi-objective SPL models require approximate techniques.

We were surprised that we did not observe a case where the exact technique worked, but where the approximate technique was significantly faster. The cost of parameter tuning for IBEA means that for models where the GIA completes within a couple of hours it is probably the better choice. Our results show that IBEA can compute very good answers for models with less than 45 features, but it does so at a comparable computation cost to the exact GIA.

Our results with IBEA are consistent with previous studies: IBEA can perform well for SPL models (Sayyad *et al.* [29]); and IBEA needs parameter tuning (Hadka & Reed [13]). We extend the results of Sayyad *et al.* [29] for using IBEA on multi-objective SPL models in four ways:

(1) a mutation rate of < 0.2 is best;

(2) a wide variety of crossover rates, population sizes, and total evaluations should be sampled;

(3) the decision to model constraint violations as an objective to be minimized should be revisited;

(4) IBEA struggles with models that have a high number of objectives, even with a small number of features (*e.g.*, ERS, with 7 objectives and 35 features).

Future work could explore a number of directions, including: improving constraint handling for IBEA; improving parameter tuning for IBEA; comparing IBEA to other MOEAs in light of various parameter tuning and constraint handling techniques (as acknowledged by Sayyad *et al.* [29]); improving scalability for GIA (*e.g.* through a parallel version of GIA); and hybrid GIA+IBEA algorithms.

It would be useful to compare the accuracy of IBEA using repair operators for constraint handling versus IBEA using an additional objective to be minimized as a way of handling constraint violations. This is important as for the largest SPL in our study (Eshop with close to 300 features) only 4% of the parameterizations of IBEA found any valid configurations at all. Moreover different constraint handling techniques might be optimal depending on whether the SPL model is highly constrained or not.

Several ways to combine GIA and IBEA can be explored. For example GIA could be used to improve the approximate Pareto front obtained by IBEA, by looking for configurations that dominate parts of it. On the other hand a partial execution of GIA could be used to generate an initial population for IBEA.

References

[1] http://www.emn.fr/z-info/choco-solver/, 2013.

[2] Andrea Arcuri and Lionel Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification & Reliability*, 2012.

[3] Andrea Arcuri and Gordon Fraser. On parameter tuning in search based software engineering. In *Proc. SSBSE*. Springer, 2011.

[4] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In *Proc. CAiSE*. Springer, 2005.

[5] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proc. VaMoS*. ACM, 2013.

[6] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[7] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS*. Springer, 2008.

[8] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., 2001.

[9] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[10] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[11] Naeem Esfahani, Sam Malek, and Kaveh Razavi. Guidearch: guiding the exploration of architectural solution space under uncertainty. In *Proc. ICSE*. IEEE, 2013.

[12] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84(12):2208–2221, 2011.

[13] David Hadka and Patrick Reed. Diagnostic assessment of search controls and failure modes in many-objective evolutionary optimization. *Evolutionary Computation*, 20(3):423–452, 2012.

[14] Mark Harman. The current state and future of search based software engineering. In *Proc. FoSE*. IEEE, 2007.

[15] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. Empirical software engineering and verification. chapter Search based software engineering: techniques, taxonomy, tutorial, pages 1–59. Springer-Verlag, Berlin, Heidelberg, 2012.

[16] John Henry Hollande. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, 1975.

[17] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., revised edition, January 2012.

[18] Kyo C Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study. Technical report, Software Engineering Institute - CMU, 1990.

[19] Ahmet Serkan Karatas, Halit Oguztuzun, and Ali H. Dogru. Mapping extended feature models to constraint logic programming over finite domains. In *Proc. SPLC*. Springer, 2010.

[20] J.D. Knowles. *Local-search and Hybrid Evolutionary Algorithms for Pareto Optimization*. PhD thesis, University of Reading, Reading, U.K., 2002.

[21] Sean Quan Lau. Domain analysis of e-commerce systems using feature-based model templates. Master's thesis, University of Waterloo, 2006.

[22] Fernando G.. Lobo, Cludio F. Lima, and Zbigniew Michalewicz. *Parameter Setting in Evolutionary Algorithms*. Springer, 1st edition, 2007.

[23] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[24] Marcílio Mendonça, Thiago Tonelli Bartolomei, and Donald Cowan. Decision-making coordination in collaborative product configuration. In *Proc. SAC*. ACM, 2008.

[25] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. Modelling and Optimization of Quality Attributes in Variability-Rich Software. In *NFPinDSML Workshop at MODELS Conference*, 2012.

[26] Derek Rayside, H.-Christian Estler, and Daniel Jackson. A Guided Improvement Algorithm for Exact, General Purpose, Many-Objective Combinatorial Optimization. Technical Report MIT-CSAIL-TR-2009-033, MIT Computer Science and Artificial Intelligence Laboratory, 2009.

[27] Pooya Saadatpanah, Michalis Famelis, Jan Gorzny, Nathan Robinson, Marsha Chechik, and Rick Salay. Comparing the effectiveness of reasoning formalisms for partial models. In *Proc. MoDeVVa*. ACM, 2012.

[28] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global Sensitivity Analysis: The Primer*. Wiley, 2008.

[29] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proc. ICSE*. IEEE, 2013.

[30] Norbert Siegmund, Marko Rosenmuller, Martin Kuhlemann, Christian Kastner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 1(3):1–31, 2011.

[31] Willard Simmons. *A Framework for Decision Support in Systems Architecting*. PhD thesis, Aeronautics & Astronautics, Massachusetts Institute of Technology, 2008.

[32] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computations, and Application*. John Wiley & Sons, Inc., 1986.

[33] David Allen Van Veldhuizen. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. PhD thesis, Air Force Institute of Technology, 1999.

[34] A. Vargha and H. D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal on Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[35] Tobias Wagner, Nicola Beume, and Boris Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Proc. EMO*. Springer, 2007.

[36] Jules White, Brian Dougherty, and Douglas C. Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, 82(8):1268–1284, 2009.

[37] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[38] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, ETH Zurich, Switzerland, 1999.

[39] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto evolutionary algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[40] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *Proc. PPSN*. Springer, 2004.

Appendix A

Randomly Generated Values for Attributes of Web Portal and Eshop feature models

A.1 Web Portal

Feature	Cost	Used Before	Defects
web_portal	7.640898996	T	5
add_services	7.509205377	T	6
site_stats	14.68939813	F	0
basic	6.166499394	T	8
advanced	10.41903914	T	5
site_search	9.758947026	T	6
images	8.148236815	T	5
text	13.46511175	T	6
html	10.95267856	T	4
dynamic	8.863099463	T	4
ad_server	7.749645779	T	6
reports	6.474565104	T	4
popups	11.39741596	F	0
banners	7.363245163	T	2
ban_img	13.42838449	T	4

ban_flash	5.452286608	T	5
keyword	6.100473521	T	5
web_server	12.32222489	T	1
logging	5.653699201	F	0
db	10.30831746	T	5
file	5.901282665	F	0
protocol	11.54075937	T	7
nttp	13.68264483	T	6
ftp	13.00885383	T	6
https	10.048601	F	0
cont	9.733988067	F	0
static	11.10162173	T	3
active	12.247117	F	0
asp	8.620782055	F	0
php	10.64190533	T	3
jsp	12.95291397	T	6
cgi	12.14168195	T	5
persistence	10.47466276	F	0
xml	14.05417168	F	0
database	6.737760508	T	5
ri	5.013527623	F	0
data_storage	9.556632783	T	3
data_transfer	5.222981863	T	4
user_auth	12.23424193	T	4
performance	13.70663109	F	0
ms	11.69030892	F	0
sec	9.089787558	T	6
min	8.283453001	T	6

A.2 Eshop

Feature	**Cost**	**Used Before**	**Defects**
eShop	7.640898996	TRUE	3
store_front	7.509205377	FALSE	0

homepage	14.68939813	TRUE	5
_id_1	6.166499394	FALSE	0
_id_2	10.41903914	FALSE	0
_id_3	9.758947026	TRUE	3
_id_5	8.148236815	FALSE	0
special_offers	13.46511175	TRUE	5
_id_6	10.95267856	TRUE	5
_id_8	8.863099463	FALSE	0
_id_9	7.749645779	TRUE	6
registration	6.474565104	TRUE	8
registration_enforcement	11.39741596	FALSE	0
_id_11	7.363245163	TRUE	6
register_to_buy	13.42838449	FALSE	0
_id_12	5.452286608	TRUE	5
_id_13	6.100473521	FALSE	0
_id_14	12.32222489	TRUE	6
shipping_address	5.653699201	FALSE	0
_id_15	10.30831746	FALSE	0
_id_16	5.901282665	FALSE	0
_id_17	11.54075937	FALSE	0
_id_18	13.68264483	TRUE	6
_id_19	13.00885383	FALSE	0
_id_20	10.048601	TRUE	5
_id_21	9.733988067	FALSE	0
_id_22	11.10162173	FALSE	0
_id_23	12.247117	TRUE	7
_id_25	8.620782055	FALSE	0
_id_26	10.64190533	FALSE	0
_id_27	12.95291397	TRUE	4
_id_28	12.14168195	TRUE	7
_id_29	10.47466276	FALSE	0
preferences	14.05417168	TRUE	6
_id_31	6.737760508	FALSE	0
_id_32	5.013527623	FALSE	0
_id_33	9.556632783	FALSE	0
_id_34	5.222981863	TRUE	5

quick_checkout_profile	12.23424193	FALSE	0
_id_35	13.70663109	FALSE	0
user_behaviour_tracking_info	11.69030892	FALSE	0
catalog	9.089787558	FALSE	0
product_information	8.283453001	FALSE	0
product_type	13.82254884	FALSE	0
eletronic_goods	13.44840831	FALSE	0
physical_goods	6.678736375	TRUE	6
services	12.67406591	TRUE	5
basic_information	11.45747072	FALSE	0
detailed_information	12.49404411	FALSE	0
warranty_information	14.1585748	FALSE	0
customer_reviews	13.06245311	TRUE	7
associated_assets	11.70111524	TRUE	6
_id_38	11.54706119	FALSE	0
_id_39	9.453300186	TRUE	7
_id_41	10.20846559	FALSE	0
_id_43	14.77407445	FALSE	0
_id_44	10.0855901	FALSE	0
_id_45	10.21091614	TRUE	3
_id_46	7.543934937	TRUE	4
_id_47	11.97887792	FALSE	0
_id_48	5.497550409	TRUE	8
_id_49	11.20808822	TRUE	5
_id_50	12.50812358	FALSE	0
product_variants	6.471631631	FALSE	0
_id_51	13.96965525	FALSE	0
size	7.448026706	FALSE	0
weight	13.51927726	TRUE	5
availability	10.50430781	FALSE	0
custom_fields	11.73915374	TRUE	4
categories	7.013436191	TRUE	5
categories_catalog	8.404022679	TRUE	5
_id_52	10.62694279	FALSE	0
_id_53	11.05043773	TRUE	5
_id_54	13.89161796	FALSE	0

_id_55	9.591618892	TRUE	2
_id_56	12.36668486	FALSE	0
_id_58	9.48789764	TRUE	4
_id_59	13.70277993	TRUE	5
_id_60	5.458508566	TRUE	7
_id_61	6.467647716	TRUE	4
category_page	12.48283554	TRUE	3
_id_62	8.034771162	TRUE	4
_id_63	6.991322414	FALSE	0
_id_65	11.51332141	TRUE	6
_id_66	10.07835819	TRUE	8
_id_67	10.85951119	TRUE	5
_id_68	14.68298238	FALSE	0
_id_69	12.86428822	TRUE	6
_id_70	12.96948842	FALSE	0
_id_71	11.50712546	FALSE	0
_id_72	10.96714539	FALSE	0
wish_list	5.550831042	FALSE	0
wish_list_saved_after_session	7.445684603	FALSE	0
email_wish_list	11.60295167	TRUE	3
_id_73	14.06034084	FALSE	0
permissions	8.326253343	FALSE	0
_id_75	12.45600556	TRUE	5
_id_76	8.594488385	FALSE	0
_id_77	12.62468952	FALSE	0
buy_paths	10.16498957	TRUE	6
_id_78	6.529268726	FALSE	0
_id_79	8.15573229	FALSE	0
_id_80	5.725949717	FALSE	0
_id_81	6.926431469	TRUE	5
_id_82	12.59760005	TRUE	5
_id_83	6.198151931	TRUE	3
_id_84	11.74649147	FALSE	0
registered_checkout	5.326602473	TRUE	4
quick_checkout	11.02629426	TRUE	4
_id_86	11.46957524	FALSE	0

_id_87	10.44279442	TRUE	4
shipping_options	14.52369267	TRUE	4
_id_88	7.075733586	TRUE	5
_id_89	6.373772515	TRUE	7
_id_90	13.16885452	FALSE	0
_id_91	8.95245234	FALSE	0
_id_92	9.743824553	FALSE	0
_id_93	13.71202363	TRUE	6
_id_95	8.050203855	FALSE	0
_id_96	5.767914527	TRUE	6
_id_98	9.514361895	FALSE	0
_id_99	5.497472803	FALSE	0
_id_100	11.14207215	FALSE	0
_id_101	9.948847677	TRUE	5
shipping_2	6.082961504	TRUE	5
_id_102	6.527724569	FALSE	0
_id_103	11.20320681	FALSE	0
_id_105	14.23962411	FALSE	0
_id_106	7.605837234	FALSE	0
_id_107	14.77738026	FALSE	0
_id_108	5.868448477	TRUE	5
_id_110	14.06501262	TRUE	4
_id_111	13.26452403	FALSE	0
_id_112	6.491361097	FALSE	0
_id_114	10.09771586	FALSE	0
_id_115	7.619948147	FALSE	0
_id_116	11.2844424	TRUE	4
_id_117	12.80394418	FALSE	0
_id_118	11.17713166	FALSE	0
_id_120	10.37514188	FALSE	0
_id_121	10.20307457	FALSE	0
_id_122	9.129481396	FALSE	0
_id_123	10.85470745	FALSE	0
_id_124	10.08707833	TRUE	6
_id_125	14.25352696	FALSE	0
_id_126	12.72480565	TRUE	6

_id_127	9.390004399	FALSE	0
_id_128	10.18228839	FALSE	0
_id_129	12.33661468	TRUE	6
_id_130	13.53337458	TRUE	4
_id_132	9.783003239	TRUE	8
_id_133	13.37500628	FALSE	0
_id_134	6.257091669	TRUE	4
_id_135	14.2828757	FALSE	0
_id_136	13.48356147	FALSE	0
_id_137	11.61632838	TRUE	4
_id_138	10.70066433	FALSE	0
_id_139	7.846548015	TRUE	6
_id_141	10.50601265	TRUE	5
_id_142	14.53524242	TRUE	5
_id_143	8.27648983	FALSE	0
_id_144	14.49041989	FALSE	0
buy_paths_288_289	5.874922984	FALSE	0
buy_paths_288_289_290	7.303193782	FALSE	0
buy_paths_288_289_291	7.308405931	TRUE	7
customer_service	9.813824236	TRUE	5
_id_146	13.67139298	TRUE	5
_id_147	9.161126272	FALSE	0
_id_148	11.363851	TRUE	5
_id_149	6.180057473	FALSE	0
_id_150	5.519473766	FALSE	0
_id_152	14.45465197	TRUE	3
_id_153	9.000032433	TRUE	6
_id_154	9.563743082	FALSE	0
_id_155	9.603053342	FALSE	0
_id_156	9.603156849	FALSE	0
_id_158	6.202492137	FALSE	0
_id_159	6.422839262	TRUE	2
user_behaviour_tracking	9.003503856	TRUE	3
_id_160	10.18170376	FALSE	0
locally_visited_pages	13.70299466	TRUE	4
external_referring_pages	8.7789141	FALSE	0

behaviour_tracked_previous_purchases	14.80397202	TRUE	6
business_management	5.235859855	FALSE	0
_id_162	10.69523188	TRUE	5
_id_163	8.343846613	TRUE	5
physical_goods_fulfillment	7.379857381	TRUE	8
warehouse_management	5.212683489	FALSE	0
shipping	13.92705447	TRUE	4
_id_166	14.31693161	FALSE	0
_id_167	5.226818865	FALSE	0
_id_168	10.74195024	FALSE	0
_id_169	14.92768997	TRUE	5
_id_171	14.6171411	FALSE	0
_id_172	12.27771866	FALSE	0
_id_173	7.728433902	TRUE	5
_id_174	8.00884347	FALSE	0
_id_175	7.470519879	TRUE	4
_id_177	14.1197523	FALSE	0
_id_178	13.16962224	FALSE	0
_id_179	7.434052405	TRUE	8
_id_180	7.660526899	TRUE	5
_id_181	6.386028029	FALSE	0
eletronic_goods_fulfillment	12.0959045	FALSE	0
_id_182	12.86065169	TRUE	4
_id_183	7.773943812	FALSE	0
services_fulfillment	7.835798718	FALSE	0
_id_184	12.5405914	FALSE	0
_id_185	5.289227421	FALSE	0
_id_186	14.570773	TRUE	5
_id_187	6.367356999	TRUE	3
customer_preferences	9.154310024	TRUE	6
_id_189	12.75135125	TRUE	4
_id_190	8.68253869	FALSE	0
targeting_criteria_previous_purchases	6.929546813	FALSE	0
_id_191	9.055213666	TRUE	5
wish_list_content	11.22372744	TRUE	7
previously_visited_pages	7.290506714	TRUE	5

_id_192	5.79535238	FALSE	0
_id_193	9.135713507	TRUE	4
_id_194	11.10953113	FALSE	0
_id_196	12.46607151	TRUE	7
_id_197	13.44724318	TRUE	5
_id_199	6.551331393	FALSE	0
_id_200	13.31522426	TRUE	8
_id_201	5.924711845	TRUE	2
_id_203	5.518117944	TRUE	6
_id_204	10.4038325	TRUE	7
_id_205	10.57646989	TRUE	7
_id_206	5.37137689	TRUE	6
_id_207	14.31502485	TRUE	4
discounts	7.830326937	FALSE	0
_id_208	6.877300316	TRUE	6
_id_209	7.118092394	FALSE	0
_id_210	12.59257361	FALSE	0
_id_211	7.580714012	FALSE	0
_id_212	9.7123076	TRUE	3
_id_214	5.792393473	TRUE	5
_id_215	14.03710454	TRUE	6
_id_216	13.29654719	TRUE	8
_id_217	5.63126988	FALSE	0
_id_218	11.68950869	TRUE	6
_id_219	13.59755799	TRUE	6
_id_220	7.625179961	FALSE	0
_id_222	5.68796352	FALSE	0
_id_223	7.800091605	FALSE	0
_id_224	8.912673728	TRUE	6
_id_225	12.1412932	TRUE	5
_id_226	10.81284532	TRUE	6
_id_228	10.81594781	FALSE	0
_id_229	8.428147387	TRUE	1
_id_230	14.49474053	FALSE	0
_id_231	8.651026317	FALSE	0
_id_232	8.755357522	FALSE	0

_id_233	11.85799398	FALSE	0
_id_235	12.04737272	FALSE	0
_id_236	11.15659244	FALSE	0
_id_237	9.962261314	FALSE	0
personalized_emails	9.232374202	TRUE	5
_id_238	13.86637173	FALSE	0
_id_239	9.167755017	TRUE	6
_id_240	7.98212522	FALSE	0
_id_241	10.76244707	TRUE	6
_id_242	11.69056185	FALSE	0
inventory_tracking	7.640803318	TRUE	3
_id_243	8.3123986	TRUE	8
procurement	8.632823657	TRUE	3
_id_244	10.73901371	TRUE	2
_id_245	12.20884754	FALSE	0
automatic	6.248853928	TRUE	4
_id_246	5.763206411	TRUE	5
reporting_and_analysis	8.942313675	TRUE	5
_id_247	8.350060139	FALSE	0
_id_248	8.118225697	FALSE	0
_id_249	13.87237119	TRUE	7
_id_250	6.050023565	FALSE	0
fulfillment_system	7.12461556	TRUE	7
_id_252	7.293364159	FALSE	0
procurement_system	10.14799509	TRUE	8
_id_253	5.047778792	TRUE	6
_id_254	8.78545664	FALSE	0
_id_255	12.15180823	FALSE	0
_id_256	10.80156896	FALSE	0
_id_257	12.70615066	TRUE	7
_id_258	5.910254713	TRUE	3
_id_259	7.485389165	TRUE	5
_id_260	12.74926436	TRUE	5
_id_261	5.109649705	FALSE	0
_id_262	13.43773911	FALSE	0
_id_263	11.96250562	FALSE	0