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

Rafael Olaechea, Derek Rayside, Jianmei Guo, Krzysztof Czarnecki
University of Waterloo
Waterloo, Ontario
{rolaechea, gjm,kczarnec}@gsd.uwaterloo.ca, {drayside}@uwaterloo.ca

ABSTRACT

Software product lines (SPLs) allow stakeholders to manage product variants in a systematical way and derive variants by selecting features. Finding a desirable variant is often difficult, due to the huge configuration space and usually conflicting objectives (*e.g.*, lower cost and higher performance). This scenario can be characterized as a multi-objective optimization problem applied to SPLs. We address the problem using an exact and an approximate algorithm and compare their accuracy, time consumption, scalability, 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 setting for acceptable approximation which can be ameliorated with known good parameter ranges. 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.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

Keywords

Software Product Lines, Multi-Objective Optimization

1. INTRODUCTION

Variability is ubiquitous. Physical products, such as automobiles and mobile phones, are produced as a set of variants, and so does the software embedded in them. *Software Product Line (SPL)* engineering is gaining momentum in academia and industry to effectively manage product variants derived from a range of configurable software assets [5].

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 [21, 4]. Stakeholders are interested in a product's quality attributes, such as weight, cost, and performance. However, feature models do not directly model the quality attributes of a product. Hence a feature model can be extended into an *attributed feature model* to describe the contribution of each feature to each quality attribute [3].

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, the quality attributes of the configuration can be measured or reliably estimated. Moreover, the configuration space of an SPL grows exponentially with the number of features. This growth makes it challenging to explore the configuration space. Furthermore, when deriving a desirable configuration, we encounter conflicting objectives, *e.g.*, lower cost and higher performance. Hence, engineers have to make tradeoffs between these conflicting objectives to search for an optimal configuration.

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 functionality. 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 sensitivity to parameter settings and 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 tradeoff between the resources and time required by exact approaches versus the difficulty in searching for appropriate parameters and 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 [31]. However, the impact of different parameter settings on the effectiveness of this algorithm was not explored. Although Arcuri *et al.* [2] showed the sensitivity of Evolutionary Algorithms to parameter settings in the context of search based software engineering, there is 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 [6, 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 paper, 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 show that GIA can produce all exact optimal solutions in less than two hours for small SPLs with up to 44 features, while IBEA can produce approximate solutions with an average accuracy of at least 42% in less than 20 minutes even for larger SPLs with 290 features. Our work is the first to implement exact multi-objective optimization for SPL configuration without weighted sum of objectives. We demonstrate the feasibility of exact algorithms for small-scale SPL multi-objective optimization problems and confirm the advantages of approximate algorithms for larger problems.
- We conduct a parameter sweep to systematically analyze the sensitivity of IBEA to its parameter settings, following the guidance from Hadka & Reed [14]. Our empirical results show that IBEA, at least without SPL specific constraint-handling techniques, 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. We also confirm that high accuracy can be obtained with mutation rates lower than 0.05.
- Our empirical study helps stakeholders understand the tradeoff 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 depending on the number of features, number of objectives and computational resources available.

2. RELATED WORK

We review some of the uses of single and multiple objective optimization in the context of SPLs. All of this work, including ours, can be considered under the umbrella of Search Based Software Engineering [15]. Much of this work uses genetic algorithms that have tunable parameters. We discuss some work on parameter tuning and sensitivity.

2.1 Optimization for SPL Testing

Johansen *et al.* [19] adapted combinatorial testing for SPLs, considering only combinations that respect the SPL configuration constraints. They further [20] improved the scalability of their algorithm, and showed that it could generate a covering array for pair-wise interaction testing for realistic feature models with up to approximately 6000 features (*e.g.* the Linux Kernel) and triple-wise testing for up to 1400 features.

Hernard *et al.* [16] used an ad-hoc evolutionary algorithm to generate products to test an SPL, such that the test suite maximizes pairwise coverage and minimizes the number of products to test and the cost of testing. They showed that their algorithm obtained better values for cost and number of products than a random test suite with equivalent pairwise coverage (similarly for the number of products). Their algorithm used a SAT solver to generate the initial population of valid products.

Outside the SPL context, Yoo *et al.* [38, 37] generated Pareto-optimal test suites considering code coverage, the number of past faults detected, and execution time.

2.2 Exact Optimization for SPL Configuration

Exact, single-objective optimization has been applied to SPL configurations by a number of authors. For example, Benavides *et al.* [3] used Constraint Satisfaction Programming (CSP) to optimize resource constraints. Karatas *et al.* [22] further introduced a mapping from attributed feature models to constraint logic programming over finite domains.

To the best of our knowledge, our work is the first that evaluates an exact *multi*-objective optimization of SPLs. In previous work [29], we implemented, but did not evaluate, GIA for multi-objective SPL configuration using MiniSAT. Here, we have reimplemented GIA with the Z3 SMT solver.

2.3 Approximate Optimization for SPLs

Approximations are often used for multi-objective optimization of SPLs because it is an NP-hard problem [36].

White et al. [36] proposed Filtered Cartesian Flattening to transform the SPL configuration problem into the multi-dimensional, multi-choice knapsack problem, to which they applied known approximation techniques.

Guo et al. [12] proposed and evaluated a genetic algorithms for SPL configuration. They reduced multiple objectives to a single objective with weightings, and handled constraint violations with a repair operation. Their evaluation does not compare the results of the genetic algorithm to the exact answers, whereas our evaluation does.

Sayyad *et al.* [31] experimented with five MOEAs for multi-objective optimization of SPL configurations. They concluded that IBEA scales best of the five when increasing the number of objectives. However, they did not systematically evaluate IBEA over different parameter settings. In contrast, we perform Sobol sampling to evaluate the sensitivity of IBEA in a comprehensive way. Moreover, we compared IBEA to an exact technique.

Sayyad *et al.* [30] subsequently used the Z3 SMT solver to generate the initial population for IBEA and they also fixed mandatory features. This hybrid approach produced better approximations of the Pareto front and improved scalability. They did not evaluate it against the exact Pareto front.

2.4 Parameter Sensitivity and Sobol Sampling

Most MOEAs have a number of adjustable parameters, such as crossover rate, mutation rate, selection strategy, population size, *etc.* In a broad study of multi-objective evolutionary algorithms, Hadka & Reed [14] demonstrated that most MOEAs, including IBEA, are sensitive to these parameter values: that is, the accuracy of the computed results varies depending on the parameters.

In a more focused study of single-objective evolutionary algorithms for test-case generation, Arcuri *et al.* [2] found that parameter settings can strongly affect results, but that the default values for parameters were quite good. They did not consider mutation rate as a parameter.

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, Stratified sampling, and Sobol sampling [28]. Sobol sampling explicitly minimizes the density differences across samples. It is now the preferred parameter sweep technique for sensitivity analysis [28], and it is what we use in this study.

3. PRELIMINARIES

3.1 Attributed Feature Models

Figure 1 presents an attributed feature model of a mobile phone platform. The model defines a set of features, including mandatory (filled circle), optional (empty circle), and alternative (arc) ones. 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.

A given feature can be assigned one or more quality attributes with values to quantify the impact the feature has on the respective qualities of a product variant, such as memory and cost. The impact each feature has on a quality attribute can be obtained through systematic measurement of different variants (*e.g.* [32, 13]). For example, in Figure 1, the impact of feature PasswordProtection on memory consumption is quantified as 20. In most of the sample models, we compute the quality of a product variant by summing up the contributions of each feature present in the variant. Some of the models also involve more complex computations, including multiplicative terms (e.g., for computing reliability), and also terms that represent feature interactions [32]. The objective functions are then to either minimize or maximize the quality attribute of a product variant.

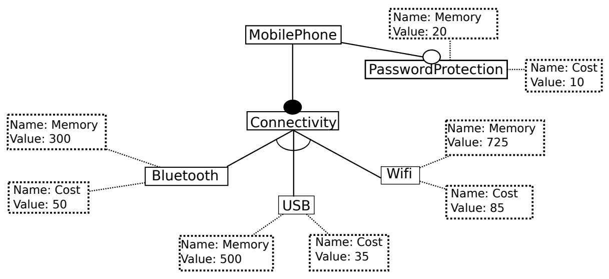


Figure 1: An attributed feature model of a mobile phone platform (adapted from [3])

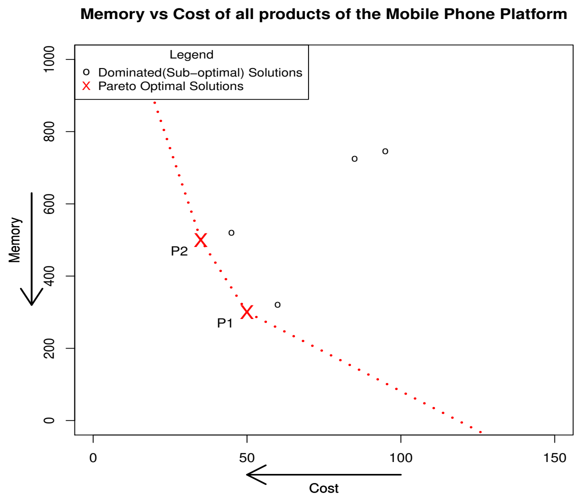


Figure 2: The objective space and Pareto front of the mobile phone platform example in Figure 1

3.2 Multi-Objective Optimization (MOOP)

Multi-objective optimization arises when optimal solutions involve trade-offs between two or more conflicting objectives. For example, Figure 2 presents the objective space of the mobile phone platform from Figure 1. This space is formed by the two quality attributes: memory consumption and cost.

Stakeholders intend to minimize memory consumption and 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* [34]. 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 illustrates six valid configurations of the mobile phone platform example in Figure 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. All Pareto-optimal solutions constitute the *Pareto front* (dotted line in Figure 2) of optimal products.

4. EXACT MOOP: GIA

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 SMT solver Z3 [6], due to its outstanding performance in the reasoning and checking of model properties [27]. GIA works as follows: on each step a candidate solution is replaced by another solution that dominates it, if one exists; the candidate solution is excluded from the search by adding a constraint. When no dominating solution can be found, the current candidate solutions is added to the Pareto front and the process is restarted. We have used exact solutions computed by our implementation of GIA to evaluate the accuracy of the IBEA heuristic (introduced in the next section).

The feature model constraints are modelled by a translation into propositional logic. The objectives are modelled using either integer or floating point variables and sums or multiplications among the contributions of each feature.

5. APPROXIMATE MOOP

The most common way to compute approximate solutions to a multi-objective optimization problem is to use an evolutionary (genetic) algorithm. Multi-Objective Evolutionary Algorithms (MOEAs) use the ideas of natural selection and evolution from nature to perform computation [9, p. 14-35]. An evolutionary algorithm consists of: *1)* an encoding for individual candidate solutions *2)* a crossover and mutation mechanism *3)* a survivor selection mechanism *4)* a parent selection mechanism *5)* an initialization mechanism and *6)* termination conditions.

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 [17, p. 70-72]. For example, for the mobile phone platform from Figure 1, each configuration would be represented by a string of six bits (as it has 6 features). This bit-string encoding was also used by Sayyad *et al.* [31].

One of the main design decisions when using a MOEA in the SPL domain is how to handle constraints. Eiben [9, p. 205-219] describes 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; (2) creating a repair operator to ensure every candidate solution is repaired to satisfy constraints; and (3) modifying the combination and mutation operators so that only valid candidate solutions are generated. Sayyad *et al.* [31] use the first approach (*i.e.* by adding as an objective to minimize the number of constraint violations) in their study of MOEAs for SPLs, and we follow them. 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. At the end of the MOEA run we filter out all invalid configurations (*i.e.* hypervolume reached is computed only over the valid configurations).

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 linear in the size of the population (*i.e.*, $c = p$). Thus, $b = c \times g$, which usually equals $p \times \left(\frac{e}{p}\right) = e$.

5.1 Indicator-Based Evolutionary Algorithm

The Indicator-Based Evolutionary Algorithm (IBEA) [40] was designed for multi-objective optimization by incorporating the *hypervolume* concept (§5.2) into its survivor selection mechanism. Independent studies by Hadka & Reed [14] and by Sayyad *et al.* [31] have confirmed that it is one of the best evolutionary algorithms for multi-objective optimization, including multi-objective optimization for SPLs.

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.*, $c = p^2$, rather than the usual $c = p$. This increase in computational cost per generation is due to computing the hypervolume indicators. Therefore, the computational budget of IBEA is characterized by:

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

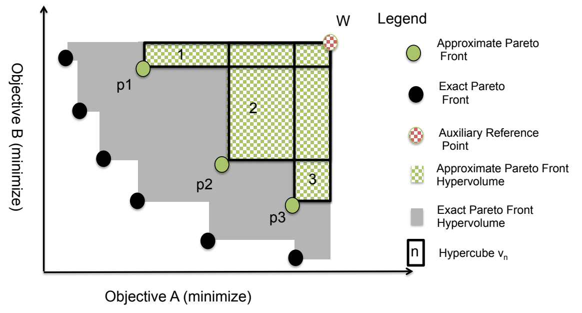


Figure 3: The hypervolume for an approximate Pareto front. We show the hypervolume for an approximate Pareto front that consists of the non-dominated solutions p1, p2 and p3. It is the union of the 2-dimensional hypercubes (*i.e.*, rectangles) v_1, v_2, and v_3, that are formed between p_i and the reference point W. The reference point W represents the worst possible value for each objective.

5.2 Accuracy Metrics

There are a variety of metrics available to measure Pareto front approximations [39], [7, p. 306-324]. Each metric characterizes the approximation's accuracy differently. We use the *Hypervolume Ratio* and *Coverage* metrics.

The *two-sets Coverage* metric is the number of exact Pareto points included in the approximation [41, 23]. Given an exact Pareto front P and an approximate Pareto front A, the Coverage of A to P is defined as follows:

$$Coverage_P(A) = \frac{|P \cap A|}{|P|} \quad (1)$$

The *Hypervolume Ratio* [35, 41] 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 3 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 metric is the ratio of these.

The hypervolume of a set of solutions Q is calculated as follows [7, p. 318]. For each solution $q_i \in Q$, a 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. In Figure 3, we wish to minimize both objectives, so W is in the upper right corner. The overall hypervolume V is the volume of the union of all such hypercubes v_i.

6. EVALUATION

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: How sensitive is IBEA to its parameter settings?
RQ 2: How fast is IBEA for acceptable accuracy?
RQ 3: What are good parameter ranges for IBEA?
RQ 4: How scalable is GIA?
RQ 5: When is it preferable to use GIA or IBEA?

6.1 Subject Models

We collected a set of attributed feature models from the recent SPL literature (Table 1), plus one from another domain (Apollo). These models cover a range of sizes, from a small one such as Berkeley DB, to large one with hundreds of features such as Eshop, or many objectives, such as ERS.

	#Features	*#CTC*	*#Objectives*
Berkeley DB [32]	12	0	4
Apollo [33]	15	3	2
ERS [10]	35	2	**7**
Web Portal [31, 25]	44	6	4
Eshop [31, 24]	**290**	**19**	4

Table 1: List of subject models. 'CTC'—Cross-Tree Constraints. Largest numbers emphasized. IBEA used as an additional objective to minimize the number of constraint violations.

Berkeley DB [32] 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.

Apollo [33] 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). This Apollo model is used with GIA only.

ERS [10] 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.

Web Portal [25] and Eshop [24] are feature models describing a product line for web portals and for e-commerce web sites, respectively. Sayyad *et al.* [31] 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.

6.2 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.

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. Although low mutation are recommended we decided to sample the whole parameter space as done by Hadka & Reed [14] to be able to more fully analyze the effect of the mutation rate.

As discussed above (§5.1), 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.

The implementation of the IBEA algorithm we used was from the JMetal framework [8] (the same implementation used by Sayyad *et al.* [31]).

6.3 Results

Here we present the results of our experiments in reference to the five research questions described above.

6.3.1 *How sensitive is IBEA to its parameters?*

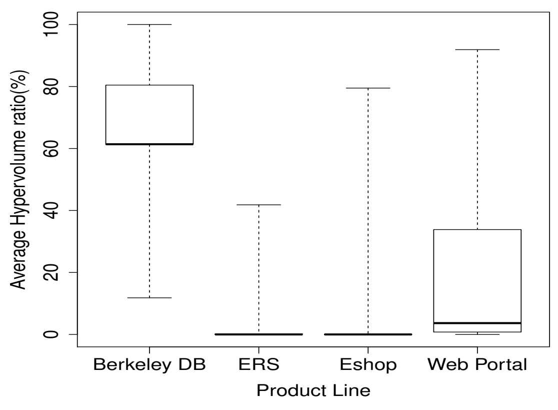
Figures 4(a) and 4(b) show box-plots of the average hypervolume and average coverage across 1000 different parameter settings for each subject model. For both ERS and Eshop, more than 75% of all parameter settings obtain an hypervolume and coverage of zero, and more than 75% of parameter settings for Web Portal produce a coverage of zero.

Furthermore, we performed the percentile ranking of the 1000 parameter settings of IBEA from the worst case to the best in terms of their obtained average hypervolume and average coverage, which is shown in Figure 5. Except the smallest case Berkeley DB, for all the other three subject models (Web Portal, ERS and Eshop), IBEA produces acceptable approximations of the Pareto front only if it happens to choose a parameter setting in a small portion (20%, 5%, and 2%,respectively) in terms of hypervolume, and a smaller portion (less than 15%, 1%, and 1%, respectively) in terms of coverage.

From Figures 4 and 5, we can observe that the accuracy of IBEA is highly sensitive to its parameter settings, which is consistent with the findings from Arcuri *et al.* [2].

6.3.2 *How fast is IBEA for acceptable accuracy?*

With the best parameter settings, IBEA can produce ac-



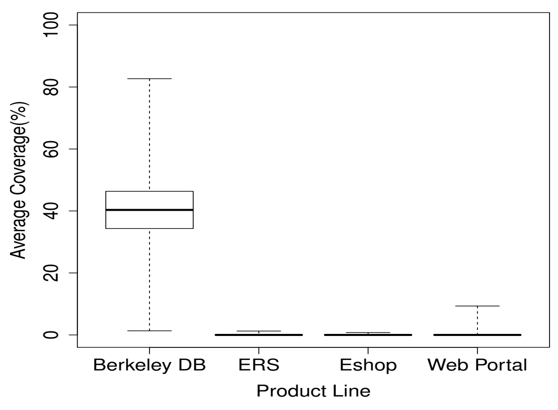
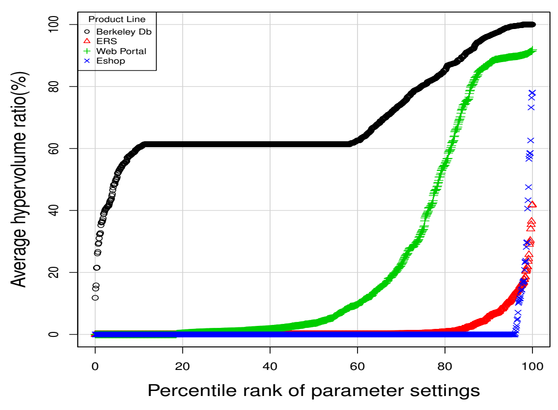


Figure 4: Average hypervolume ratio and coverage across 1000 different parameter settings. Whiskers denote the best and worst parameters.



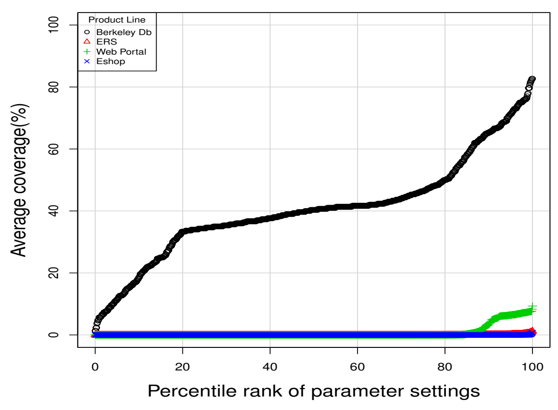


Figure 5: Average hypervolume and coverage obtained by IBEA based on percentile ranking of 1000 parameter settings

ceptable approximations of the Pareto front (>80% hypervolume) for models with 4 or fewer objectives (Figure 4, top). These solutions might even include a number of points from the exact Pareto front (Figure 4, bottom). But with poor parameter settings IBEA can also produce worthless solutions (Figure 4). How long does it take to find and execute good parameters 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 7 shows the best hypervolume obtained for a given size of Sobol sample, from 0 to 1000. We see that for Berkeley DB, the smallest and simplest model, 100% hypervolume is obtained after only 10 Sobol samples. The next most complicated model, Web Portal, gets close to its best hypervolume 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 sam-

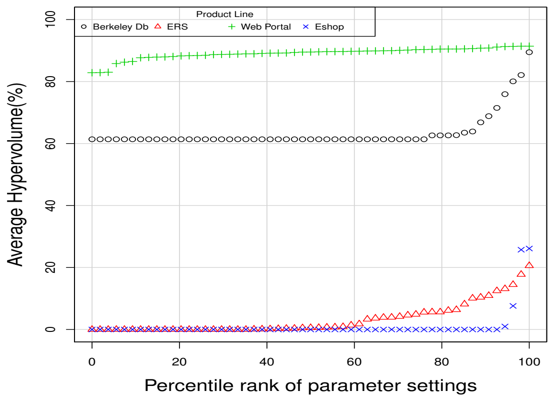


Figure 6: Average hypervolume obtained by IBEA based on percentile ranking of parameter settings with mutation rate lower than 0.05

pling only in parameter ranges that are known to be good for the given problem domain. Our experiments (§6.3.4) 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. Are there runs with similar accuracy to the best parameter settings and significantly lower run times? The answer is yes. Figure 8 groups 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). Moreover from the raw data we observe that Web Portal we obtain 88% accuracy in terms of hypervolume in under 15 seconds, and for Berkeley DB over 90% accuracy in under 5 seconds. The more complex models, ERS and Eshop, require their highest budget ranges to get their best accuracy.

In summary, IBEA can achieve good accuracy for the smallest models in a few seconds or minutes. However as either the number of features or objectives increase, larger time budgets on the order of tens of minutes to a few hours are required. The time required appears to be more a function of the number of objectives than the number of features.

6.3.3 *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 on MOEAs (the graph upon which we base this conclusion is not shown here due to space considerations). We also analyzed the accuracy of IBEA when fixing the mutation rate to be lower than 0.05, and ranked the accuracy obtained from worst to best parameter in that case (55 parameter settings) in Figure 6. Figure 6 shows that by fixing the mutation rate we can ensure IBEA obtains consistently good accuracy across the parameter space for Web Portal and Berkeley DB. Moreover within this restricted parameter space a larger proportion of parameter configurations produce non-zero accuracy for Eshop and ERS and values close to their peak accuracy are found within this restricted parameter space. Other studies of MOEAs in Software Engineering used mutation rates as high as 0.3 [11], while others as low as 0.05 [31] or even 0.001 [30].

We found accurate results at all crossover rates, with no clear trend about what values are better or worse: the entire range of crossover rates (0–1) should be sampled.

For the smaller models we found that larger budgets ($p \times e$, §5.1) produced better results (Figure 8). 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.* [31], then our results show that a variety of budget levels (*i.e.*, values of p and e) should be sampled.

6.3.4 *How scalable is GIA?*

Table 2 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.

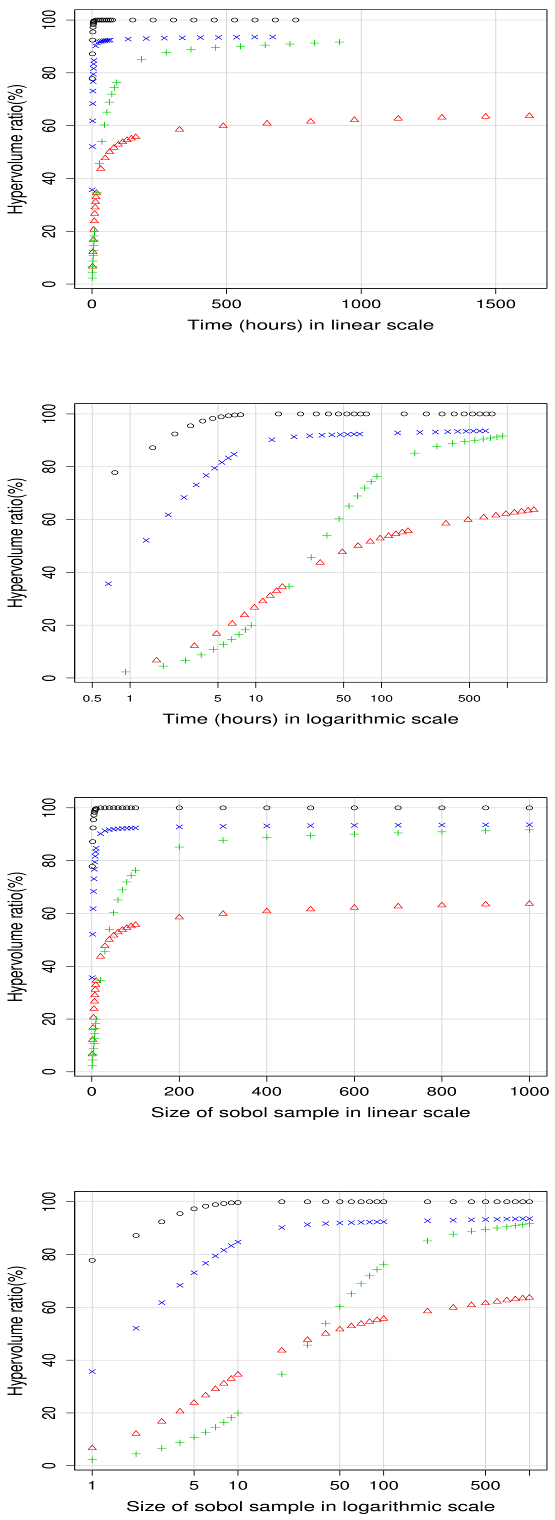


Figure 7: Maximum hypervolume ratio (%) and time required by IBEA using different size of Sobol sample of parameter settings. Berkeley DB: ○ ERS: △ Web Portal: × Eshop: +

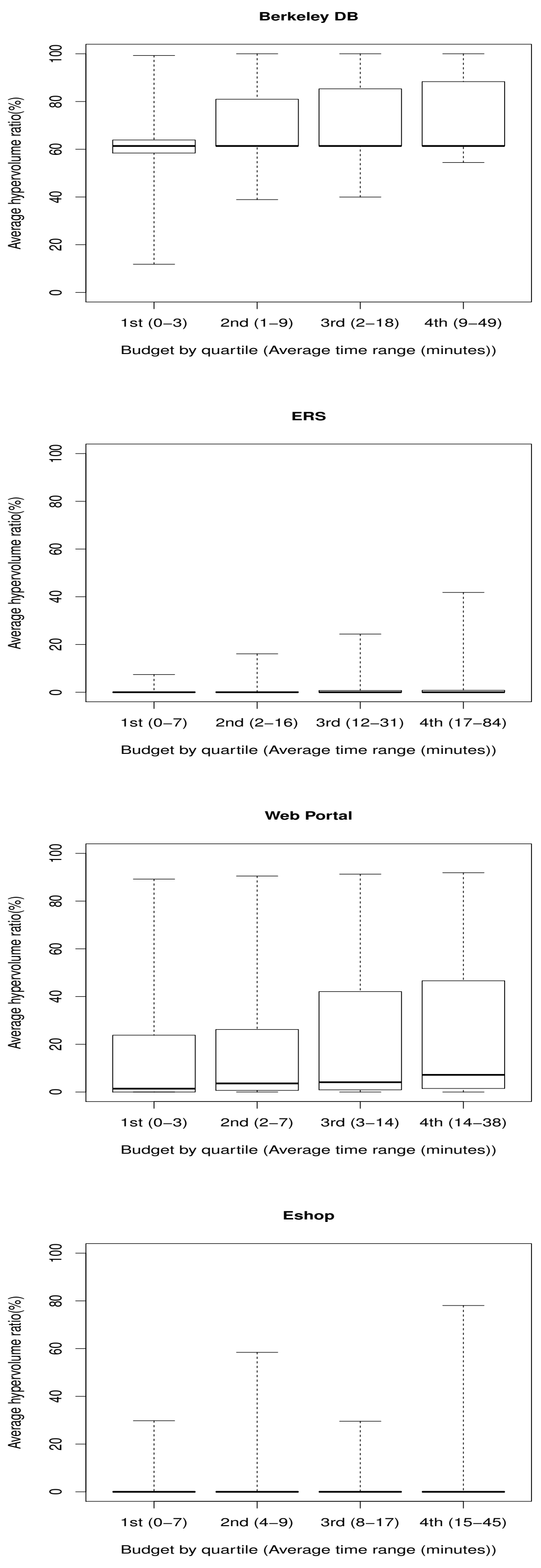


Figure 8: Average Hypervolume Ratio of IBEA in different ranges of time budget. Whiskers denote the best and the worst cases.

	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 2: Time consumption of GIA

The third column collects the mean and standard deviation of the time consumption of running GIA 1000 repetitions for each subject model.

From Table 1 and Table 2, we can see that GIA can find the Pareto front of an attributed feature model with up to 44 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 other subject models, GIA may run faster or slower than IBEA, depending on the parameter settings of IBEA.

6.3.5 *When is it preferable to use GIA or IBEA?*

For small models, such as Berkeley DB, both GIA and IBEA run quickly and produce good results (although GIA's results are better). For large models, such as Eshop, GIA will time out, and so IBEA is the only option.

In the middle it is hard to predict which algorithm will work better for a given problem. The WebPortal case study presents a result that might be expected: IBEA computes a good solution quickly (>80% of the hypervolume in a few seconds), whereas GIA computes the exact answer slowly (1.65 hours, see Table 2). The ERS case study, however, is surprising: GIA computes the exact answer in 32s (Table 2), whereas even with the best parameters we found, IBEA would spend 50 minutes to compute only 60% of the hypervolume. The high number of dimensions in ERS (7) might contribute to IBEA's slow runtime. IBEA requires computing the hypervolume, which gets more expensive as the number of objectives increases.

7. THREATS TO VALIDITY

Arcuri & Briand [1] 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 deviation of the running times we did measure is relatively low; thus, we believe that the results would be similar. 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: We would still conclude that IBEA is very sensitive to its parameters (§6.3.2), and that GIA is a better choice for ERS than IBEA is (§6.3.5). 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 (§6.3.3).

We could have produced more detailed answers for research question 1 (§6.3.1) and 3 (§6.3.3) 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 [14] used the variance decomposition method in their study.

Sayyad *et al.* [31] 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 could not 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.* [31] generated, in which case it might be difficult to compare some of the measurements in this paper with their paper.

Our estimates of how long it takes IBEA to produce accurate results (§6.3.2) were done without using a clever tuning strategy. The tuning strategy we used was Sobol sampling of known good parameter ranges (§6.3.3). Perhaps a more clever tuning strategy could produce similar results in less time, such as the one proposed by Hutter *et al.* [18]. A better tuning strategy would not, however, change our conclusion about when to use GIA and when to use IBEA (§6.3.5).

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.*'s study [31], 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 seems to be true of all of the multi-objective SPL models that we are aware of, but it does not hold in other disciplines. For example, in Civil Engineering (*e.g.*, [14]), 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. We included the Apollo [33] 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 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 [14].

A broader selection of subject models, in combination with the variance decomposition method [28], might give more precise parameter settings 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.

8. CONCLUSION AND FUTURE WORK

We have demonstrated that it is feasible to compute exact solutions for multi-objective SPL models with up to 44 features. Previously exact techniques have been applied only to single-objective SPL models (*e.g.*, [3, 22]), and it was assumed that approximate techniques were required for multi-objective models (*e.g.*, Sayyad *et al.* [31]).

Having exact solutions allowed us to perform the most accurate evaluation of an MOEA (IBEA) in the context of SPLs to date. Previous evaluations of MOEAs in the context of software engineering have had to approximate the true solutions (as we had to do for the largest SPL model Eshop).

With each of our four case studies we found a different result in terms of the runtime and accuracy of the exact (GIA) and approximate (IBEA) algorithms. Berkeley DB is a simple problem and both algorithms solve it quickly and well. At the other extreme, Eshop is still too large for our exact technique. The medium-sized WebPortal and ERS models are more interesting.

WebPortal, with 44 features and 4 objectives, solves quickly and well with IBEA: $>80\%$ of the hypervolume can be found in a few seconds simply by holding the mutation rate below 0.05. GIA computes the exact answer in over 1.5 hours.

ERS, on the other hand, takes GIA just over thirty seconds to compute the exact solution. Assuming the best parameter values that we found, it still takes IBEA almost an hour (50 minutes) to find only 60% of the hypervolume. IBEA is never able to go much beyond 60% hypervolume on this SPL model. ERS has 35 features and 7 objectives.

Our recommendation is that both IBEA and GIA should be run. In some cases both will return good solutions quickly. In other cases, GIA will timeout. In other cases, IBEA will never compute a good solution. At this time we cannot predict *a priori* which algorithm will give the best runtime/accuracy tradeoff for a given SPL model.

We found that for simple SPL models, holding the mutation rate to ≤ 0.05 was a good starting point for IBEA. For SPL models with a higher number of objectives or a higher number of features, IBEA requires more tuning. We found that mutation rates up to 0.2 sometimes worked well, that the complete range of crossover rates should be explored, and that a wide range of populations sizes and computing budgets should be sampled.

Future work could explore a number of directions, including: improving constraint handling for IBEA (perhaps a repair operator, or a better mutator/generator); improving parameter tuning for IBEA; improving scalability for GIA; and hybrid GIA+IBEA algorithms.

References

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

[2] A. Arcuri and G. Fraser. On parameter tuning in search based software engineering. In *Proc. SSBSE*, 2011.

[3] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *Proc. CAiSE*. Springer, 2005.

[4] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. A survey of variability modeling in industrial practice. In *Proc. VaMoS*, 2013.

[5] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[6] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS*. Springer, 2008.

[7] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, 2001.

[8] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

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

[10] N. Esfahani, S. Malek, and K. Razavi. Guidearch: guiding the exploration of architectural solution space under uncertainty. In *Proc. ICSE*, 2013.

[11] 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*, 2013.

[12] 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*, 84(12):2208–2221, 2011.

[13] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. Variability-aware performance prediction: A statistical learning approach. In *ASE*, 2013.

[14] D. Hadka and P. Reed. Diagnostic assessment of search controls and failure modes in many-objective evolutionary optimization. *Evol. Comput.*, 20(3):423–452, 2012.

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

[16] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Multi-objective test generation for software product lines. In *Proc. SPLC.* ACM, 2013.

[17] J. H. Hollande. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, 1975.

[18] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stutzle. Paramils an automatic algorithm configuration framework. *JAIR*, 36:267–306, October 2009.

[19] M. F. Johansen, O. Haugen, and F. Fleurey. Properties of realistic feature models make combinatorial testing of product lines feasible. In *Proc. MODELS*, 2011.

[20] M. F. Johansen, O. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *Proc. SPLC.* ACM, 2012.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study. Technical report, Software Engineering Institute - CMU, 1990.

[22] A. S. Karatas, H. Oguztuzun, and A. H. Dogru. Mapping extended feature models to constraint logic programming over finite domains. In *Proc. SPLC*, 2010.

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

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

[25] M. Mendonça, T. T. Bartolomei, and D. Cowan. Decision-making coordination in collaborative product configuration. In *Proc. SAC.* ACM, 2008.

[26] D. Rayside, H.-Christian. Estler, and D. Jackson. A Guided Improvement Algorithm for Exact, General Purpose, Many-Objective Combinatorial Optimization. Technical Report MIT-CSAIL-TR-2009-033, MIT CSAIL, 2009.

[27] P. Saadatpanah, M. Famelis, J. Gorzny, N. Robinson, M. Chechik, and R. Salay. Comparing the effectiveness of reasoning formalisms for partial models. In *Proc. MoDeVVa.* ACM, 2012.

[28] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis: The Primer.* Wiley, 2008.

[29] S. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside. Modelling and Optimization of Quality Attributes in Variability-Rich Software. In *NFPinDSML Workshop at MODELS Conference*, Oct. 2012.

[30] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *ASE*, 2013.

[31] 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 *ICSE*, 2013.

[32] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, S. Apel, and G. Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality*, 1(3):1–31, 2011.

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

[34] R. Steuer. *Multiple Criteria Optimization: Theory, Computations, and Application.* Wiley, 1986.

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

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

[37] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *Proc. ISSTA.* ACM, 2007.

[38] S. Yoo and M. Harman. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.*, 83(4):689–701, Apr. 2010.

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

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

[41] 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.