An Approach for Optimized Feature Selection in Large-scale Software Product Lines

Xiaoli Lian, Li Zhang, Jing Jiang, William Goss

PII:	S0164-1212(17)30043-2
DOI:	10.1016/j.jss.2017.02.044
Reference:	JSS 9931

To appear in:	*The Journal of Systems & Software*

Received date:	12 April 2016
Revised date:	5 November 2016
Accepted date:	24 February 2017

Please cite this article as: Xiaoli Lian, Li Zhang, Jing Jiang, William Goss, An Approach for Optimized Feature Selection in Large-scale Software Product Lines, *The Journal of Systems & Software* (2017), doi: 10.1016/j.jss.2017.02.044

An Approach for Optimized Feature Selection in Large-scale Software Product Lines

Xiaoli Lian[a], Li Zhang[a], Jing Jiang[a], William Goss[b]
{lianxiaoli, lily, jiangjing}@buaa.edu.cn, william.t.goss@gmail.com

[a]*State Key Laboratory of Software Development Environment, Beihang University, Beijing, China, 100191*
[b]*DePaul University, Chicago, IL, 60604*

Abstract

Context: Feature selection in Product Line Engineering is an essential step for individual product customization, in which the multiple objectives, that are often competing and conflicting, have to be taken into consideration. These objectives always need to be balanced during selection, leading to a process of multi-objective optimization. What's more, the massive complex dependency and constraint relationships between features present another huge challenge for optimization.
Objective: In this work, we propose a multi-objective optimization algorithm, IVEA-II, to automatically search through configurations to obtain an optimal balance between various objectives. Additionally, all the relationships between features must be conformed to by the optimal feature solutions.
Method: Firstly, a two-dimensional fitness function in our previous work is reserved. Secondly, to prevent the negative impact of this 2D fitness on the diversity of final Pareto Fronts, the crowding distance is introduced into each fitness-based selection. Lastly, a new mutation operator is designed to improve the scalability of IVEA-II.
Results: A series of experiments were conducted to verify the effectiveness of IVEA-II on five large-scale feature models with five optimization goals.
Conclusion: Experiments showed that IVEA-II can generate more valid solutions over a set period of time, with optimal solutions also having better diversity and convergence.

Keywords: Software Product Lines, Feature Selection, Product Derivation, Multi-objective Optimization

1. Introduction

Product Line Engineering (PLE) as an effective production approach has been attracting more attention both in academic and industrial domains. Many organizations, including Boeing, Bosch Group, Philips and others [1], have obtained noticeable success with PLE and have become models representing what PLE can achieve.

The process of making an individual software is called product customization (also product derivation), in which the core activity is selecting a suitable feature set which satisfies certain specified requirements. These requirements can be functional or nonfunctional, with some requirements very likely competing and even conflicting with one another. For example, when designing a mobile phone, a user would like more useful functions (i.e., more features) and less run-time errors. However, the composition of more features will likely bring a higher possibility of also creating more errors. Due to these conflicts, a balance to the objectives is critical for determining the final feature set and shows that feature selection is essentially a multi-objective optimization process. Meanwhile, the dependency and constraint relationships between features are an important concern that limits the existences of some features in various configurations. For example, the *camera* in a mobile phone always requires a *high resolution screen*. Features derived from specific functional requirements are also mandatory for product derivation, which leads to another constraint within feature selection. With numerous constraints, the search-space for optimized selections is restricted and the generation of valid solutions is increasingly difficult.

In our previous work [1], an Indicator optimization and rules Violation controlling Evolutionary Algorithm (IVEA), was proposed to do constrained multi-objective optimization towards feature selection. Specifically, the constraints and required features derived from definite functional requirements, which have to be satisfied in all configurations, were defined as rules. The multiple competing objectives were transferred as the optimization goals. A two-dimensional (2D) fitness function was proposed to control the rule violations and the performance of multi-objective optimizations, respectively. In these two dimensions, the conformance to constraints has higher priority

[1]Product Line Hall of Fame: http://www.splc.net/fame.html

than objective optimization. The underlying notion is that only valid solutions that follow all constraints make sense in practice. The most significant advantage of IVEA is that it places more emphasis on the conformance of the constraints between features, compared to the existing works [2, 3]. In addition, IVEA lowers the number of objectives within the feature selection by promoting one objective to a partial fitness function. On the basis of 2D fitness function, the common evolutionary processes, including environment selection, crossover and mutation, were adopted to select the optimal solutions. Although the significant efficiency of 2D fitness function has been shown [1], IVEA is not able to work efficiently for large feature models. Due to enormously large models, such as *Linux* with over 6000 features, scalability is a critical facet that needs to be examined. Therefore, within this paper we will explore some novel evolutionary operations.

Great work has been done for optimized feature selection, such as Sayyad et al. who firstly explored the Multi-Objective Evolutionary Algorithms (MOEAs) to select features in the face of multiple objective balancing [2]. This work showed that IBEA (Indicator-Based Evolutionary Algorithm) [4] outperformed other popular MOEAs, like NSGA-II [5] and SPEA2 [6]. This work was then expanded by exploring some simple heuristics to improve the scalability of IBEA [3]. A highly distinguished observation is that planting a *feature-rich* seed in the initial population can remarkably scale up IBEA's optimization ability, just like "a single straw" is enough to "break the camel's back". This magic seed, in the midst of the initial population, drives us to improve scalability of IVEA.

Thus, the purpose of this paper is to design IVEA-II by improving IVEA for scalability and comparing the performance against the current leading multi-objective optimization based feature selection algorithms. The main contributions of this paper can be summarized as:

- Designing a new mutation operator based on the idea of planting valid seeds in the initial population [3] to improve the scalability of IVEA.
- Improving the generation of the final optimal configurations by implementing the crowding distance from NSGA-II [5].
- Demonstrating IVEA-II's scalability through a series of experiments on five objective optimizations with five large feature models ranging from 1,244 to 6,888 features.

- Demonstrating the effects of crucial points within IVEA-II including the 2D fitness function, crowding distance and our new mutation on the generation and quality of approximate solutions.

The rest of this paper is organized as followed. Firstly, some background knowledge is introduced in Section 2, including the information about feature models (Section 2.1) and the leading algorithms in multi-objective optimization based feature selection (Section 2.2). Next, our proposed approach, IVEA-II, is presented in Section 3. After that, Section 4 defines our research questions. Section 5 and 6 detail a series of experiments that are conducted to perform the systematic evaluation to IVEA-II. Then, Section 7 discusses the potential applications of our approach, the threats to validity and the limitations of our approach. Section 8 highlights the related works in automatic feature selection. Lastly, we conclude the work in Section 9.

2. Background

2.1. Feature Models

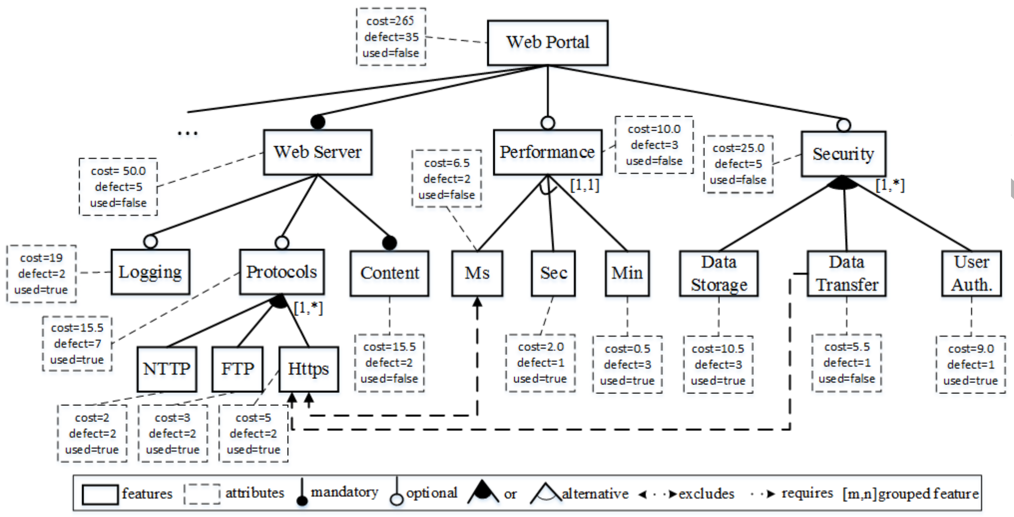
Proposed as an important part of Feature-Oriented Domain Analysis (FODA) [7], feature models (FM) are a popular way to express the all available features in a product family. A feature is defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a software or system" [7]. In short, features are logic units that are configured to form individual products. FM is a compact way to describe all features and their related constraints.

Figure 1 is a part of the Web Portal feature model [8]. We can see that all features (in rectangles) are organized in a hierarchy. There are also several types of relationships between features (solid line and dashed line with arrows), which work as constraints for the coexistence of different features in any configuration.

The relationships between features can be summarized as follows [9]:

1) Relationships between a parent and its child features, including *mandatory*, *optional*, *alternative* and *or* (solid line in Figure 1).

- *Mandatory:* If a parent is selected, the child has to be selected too. For instance, since a *mandatory* relationship exist between *Web Server* and *Content* in Figure 1, feature *Content* has to coexist with it's parent, *Web Server*.



cost=265 defect=35 used=false
Web Portal
...
Web Server
cost= 50.0 defect=5 used=false
Performance
cost=10.0 defect=3 used=false
cost=6.5 defect=2 used=false
[1,1]
cost=25.0 defect=5 used=false
Security
[1,*]
cost=19 defect=2 used=true
Logging
Protocols
Content
Ms
Sec
Min
Data Storage
Data Transfer
User Auth.
cost=15.5 defect=7 used=true
[1,*]
NTTP
FTP
Https
cost=15.5 defect=2 used=false
cost=2.0 defect=1 used=true
cost=0.5 defect=3 used=true
cost=10.5 defect=3 used=true
cost=5.5 defect=1 used=false
cost=9.0 defect=1 used=true
cost=2 defect=2 used=true
cost=3 defect=2 used=true
cost=5 defect=2 used=true
features
attributes
mandatory
optional
or
alternative
excludes
requires
[m,n]grouped feature

Figure 1: Partial Feature Model of Web Portal

- *Optional:* If a parent is selected, the child does not necessarily need to be selected, like the relationship between *Web Server* and *Logging* in Figure 1.
- *Alternative:* This group relationship is between a parent and the multiple children. Within the *Alternative* relationship, a single child can be selected if the parent has been selected. An *alternative* example is *Performance* and its children features: *Ms*, *Sec* and *Min* in Figure 1.
- *Or:* This is another type of group relationship, where any of the children can be selected when the parent is selected. In Figure 1, this relation exists between *Security* and its children: *Data Storage*, *Data Transfer* and *User Auth.* If the parent, *Security*, is selected, any child feature can be selected or not selected independent of one another.

2) Cross-tree constraints which are typically *inclusion* and *exclusion* statements (dashed line with arrows in Figure 1).

- *Inclusion:* This refers to the *requires* relationships in the FM. In Figure 1, *Data Transfer* requires *Https* which means that if feature *Data Transfer* is selected, feature *Https* must be selected as well.
- *Exclusion:* This refers to the *excludes* in the FM. In Figure 1, *Ms* excludes *Https* which means feature *Ms* and feature *Https* cannot coexist

in any configuration.

Therefore, within a software product line, some features may be part of all products; while others may not appear in any. The former are defined as *core features* and the latter as *dead features* [9]. During feature selection, the *core features* are usually developed first and the dead features are excluded.

A FM defines a set of all permitted configurations. It is usually formatted as a Boolean expression in Conjunctive Normal Form (CNF). The CNF formulation is a conjunction of several clauses, where each clause presents one constraint. For instance, feature *Data Transfer* requires *Https* can be expressed as *Data Transfer* $\rightarrow$ *Https*. The partial FM of Web Portal in Figure 1 can be described as: $(WebPortal \leftrightarrow WebServer) \wedge (WebServer \leftrightarrow Content) \wedge (Logging \rightarrow WebServer) \wedge (Protocols \rightarrow WebServer) \wedge (Protocols \rightarrow NTTP \vee FTP \vee Https) \wedge (Performance \rightarrow WebPortal) \wedge (Performance \rightarrow XOR(Ms, Sec, Min)) \wedge (Security \rightarrow WebPortal) \wedge (Security \rightarrow DataStorage \vee DataTransfer \vee UserAuth.)$. In practice, a single feature is always more conveniently denoted by a unique ID. The process of mapping feature models to CNF is a relatively straightforward process [9].

Besides the basic notations of feature model, several extensions have been proposed, such as cardinality [9] and an attributed feature model [9, 10]. Cardinality is in the form of [m,n], declaring that at least m and at most n features in a group can be selected [9]. In the attributed feature model, extra-functional information is expressed as attributes of features. In Figure 1, we add three attributes for each feature, i.e., *cost*, *defect* and *used*.

The attributes of leaf features are usually assigned directly, and the attributes of mediate features can be calculated based on that of their children [11, 12, 13]. In this work, we assume the related attributes of features have been assigned.

2.2. State of the Art Algorithms

Sayyad et al. tried to use Multi-Objective Evolutionary Algorithms (MOEAs) to do optimized feature selection and show the significance of the Indicator-Based Evolutionary Algorithm (IBEA) [4] by comparing the results with other commonly used algorithms such as Nondominated Soring Genetic Algorithm (NSGA-II) [5], Strength Pareto Evolutionary Algorithm, version2 (SPEA2) [6], Fast Pareto Genetic Algorithm (FastPGA) [14] and others. The distinguishing aspect of IBEA is that it utilizes the user's preference on quality indicators to guide the optimization process.

Based on the improved IBEA [3], Henard et al. proposed SATIBEA and illustrated the outstanding advantage of SATIBEA over IBEA with five large feature models [15]. The main points of SATIBEA is introduced as follows.

The first critical point is from the work of Sayyed et al., where they proposed adding valid solutions (i.e., seeds) into the initial population [3]. Through experiments, it was shown that the seed in initial population can considerably enhance the efficiency of generating optimal valid solutions [3]. It was also shown that one *feature-rich* seed has even more influence than 30 seeds selected randomly on the generation of valid solutions.

Another important part of the design in SATIBEA is the mutation operator. Firstly, the mutation operator finds the features that do not involve the violation of any constraints. It then removes the rest of the feature assignments which are related to constraint violations. Finally, this partial configuration is given to an SAT solver to obtain a valid configuration. To promote diversity, Henard et al. randomly permuted three parameters that work on the search for constraint-satisfying solutions within the SAT solver. A solution is then randomly picked from the population and replaced with the new valid one. In the end, these smart mutation and replacement operations improve the quality and diversity of the optimal solutions.

In the next sections, the improved IBEA and SATIBEA algorithms are utilized as baselines.

3. Our Proposed Approach: IVEA-II

The aim of this paper is to propose a new algorithm, IVEA-II, which is designed based on our previous approach, IVEA [1]. IVEA-II should be able to select approximate features from large-scale feature models under the constraints of multi-objective optimization and feature dependencies. Due to the remarkable impact of the 2D fitness function on the optimized selection process, we adopt this two-dimensional selection criteria in IVEA-II. To improve the diversity of solutions, we introduce the crowding distance from NSGA-II [5] to the environment selection. The most important design in IVEA-II's scalability is a novel mutation operator, which plants valid solutions in the mutation phase and guides the optimized selection.

3.1. Main Loop

For the sake of clarity, we firstly list the entire work flow of IVEA-II in Figure 2, in which our critical operators are signed with a '*'.

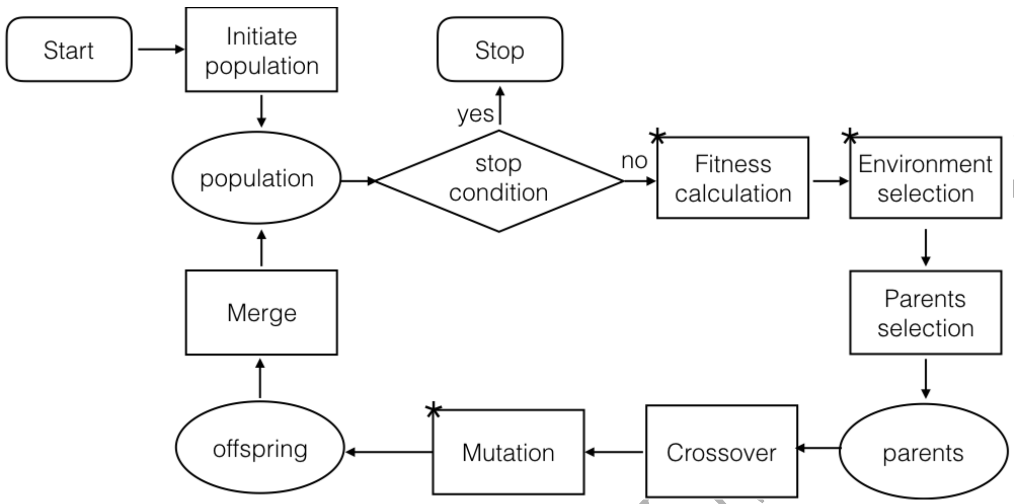


Figure 2: The Work Flow of IVEA-II

Initially, a population with size n is produced. Like in previous works [15, 3], this is not an entirely random process, during which the core features will be selected and the dead ones will be removed automatically. This process alone can save massive amounts of time during the evolutionary process.

IVEA-II then runs the fitness function on each solution to provide the basis for the following selection operations, including environment and parents selection. The operation of environment selection aims to select the fittest solutions by eliminating the ones with inferior quality. The solutions with less constraint violations and better optimization performance are preferred. During the evolution, new solutions will be created through crossover and mutation operations. This is essential for the evolutionary process and critical for ensuring a diverse population. To begin, two better solutions will be selected as parents to generate the offspring. The idea behind this is that the parents with high quality genes will be more likely to generate better offspring. More specifically, two solutions are picked at random and the dominator is selected as one parent. The dominator is voted based on the principle of Binary Tournament Selection. There are four different scenarios to decide the domination relationship: a) if the two solutions vary in the constraint violations, the solution violating less constraints is selected; b) if the two solutions violate the same number of constraints, and one solution dominates the other one in the other multiple objectives except the constraint violations, the dominator is selected as one parent; c) if the two solutions violate the same number of constraints, and these two solutions can't dominate each other, and the two solutions have different *Crowding Distance* (CD) values, the solution with bigger CD is selected as one parent; d) if two solutions violate the same number of constraints, have the same CD value, and can't dominate each other, then return one solution at random as one parent. This process is repeated again, to select one more parent.

The crossover operation then takes place with a certain probability being applied. Another way of creating new solutions is mutation, which changes the genes of individual solutions. In this phase, we add valid solutions into the entire population to promote the quality of the whole gene bank and increase the possibility of generating valid solutions. The generated offspring are merged into the population and the new circle of evolutionary process continues.

In the next sections, the improved operators will be discussed in detail.

3.2. Two-Dimensional Fitness Function

A fitness function is the core of an evolutionary algorithm, which sets the standard for selecting the fittest individuals. Originally, the fitness function in IBEA stands for a user's preference on an optimization goal defined in term of a binary indicator [1]. Traditionally, the control on constraint violations of feature configuration is treated as an ordinary objective as others [2, 3, 15]. However, it is well known that feature model conformance is the prerequisite of multi-objective optimization. Thus, we enhance the priority of constraints conformance by putting it and the multiple attributes optimization as two parallel goals, expressed as a two-dimensional fitness function.

The first dimension, named as *infeasibility* ($\mathcal{I}$), is to control the constraint violations. Previously, we only considered the evaluation time as a stop condition [1]. When the evaluation time is large enough, the *infeasibility* will be small enough to select the valid solutions. However, sometimes the running time is given which always results in small evaluation times for large-scale models. To work on larger and more complex feature models, we improve $\mathcal{I}$ by adding a regulatory factor with a value less than one, to promise that the *infeasibility* value can reach an appropriate (small) value with the increasing evaluation times. The formula is defined as

$$\mathcal{I}_e(A) = \frac{\boldsymbol{r}}{|A|} \sum_{i=0}^{|A|} violation(a_i) \tag{1}$$

with $a_i \in A$. A is the population and |A| is the cardinality. e stands for the current evaluation time. *violation(a_i)* is the number of constraints that solution a_i violates in all. The regulatory factor denoted as $\boldsymbol{r}$ is defined as

$$\mathbf{r} = \begin{cases} 1, & if\, \underset{a_i \in A}{max}\{violation(a_i)\} = 0 \\ \frac{\underset{a_i \in A}{max}\{violation(a_i)\}}{constraintNum}, & if\, \underset{a_i \in A}{max}\{violation(a_i)\} > 0 \end{cases} \quad (2)$$

Here, *constraintNum* is the total number of constraints in a feature model.

infeasibility ($\mathcal{I}$) works as a threshold in a way that during each evaluation all solutions that violate more constraints than $\mathcal{I}$ will be discarded. Therefore, the average constraint violation of the population will be decreased and as a result, the $\mathcal{I}$ turns gradually smaller. This also allows a larger threshold if a broader search space is desired. The idea behind this is to promote correctness in the results and decrease the diversity loss as much as possible.

The second dimension $\mathcal{U}$ borrowed from IBEA gives every solution a value indicating its quality in multi-objective optimization. The equation is listed as

$$\mathcal{U}(x^1) = \sum_{x^2 \in P \backslash \{x^1\}} -e^{-I(\{x^2\},\{x^1\})/k} \quad (3)$$

It is a measure for the "loss in quality" if solution x^1 is removed from the population P. $I(\bullet)$ is a dominance preserving indicator, which maps the quality comparison of two optimal solutions relatively to each other into a real number. The selection of a quality indicator depends on the users. Thus, the user's preference information on the quality has been integrated into the optimization process in this way. The commonly used indicators include $I_{\epsilon+}$ and I_{HD}. In this work, we select I_{HD} due to its significant performance on feature selection when it was utilized in IBEA, as shown in our prior paper [1]. Another reason is that this work focuses on improving our previous IVEA, which adopts I_{HD}. k is a scaling factor with a value of greater than 0. The default value is 0.05 in many implementations, such as jMetal [16].

3.3. Environment Selection

Environment selection aims at eliminating inferior results and preserving the fittest. Our environment selection is conducted based on our 2D fitness function. It is a two phase operation which begins by discarding the solutions with more violations than $\mathcal{I}$ (the first dimensional fitness value) in the current iteration. Then, the extra solutions with values worse than $\mathcal{U}$ (the second dimensional fitness value) are removed.

In both of these phases we discard one worst solution at a time. To be specific, during the first phase, if a solution has the most constraint violations and this violation number is bigger than the threshold, it will be discarded. A similar process occurs in the second phase. At times, there can be multiple candidate solutions to be removed. In our prior work, all of these candidates were abandoned [1]. We have now found that this largely reduces the search-space and is detrimental in diversifying our final optimal solutions.

We have introduced crowding distance (CD) into IVEA-II, which is a famous mechanism of NSGA-II [5] to promote the diversity of the final optimal solutions. Crowding distance gives an estimate of the density of solutions surrounding a particular solution in the population by calculating the average distance between two points in each of the objectives. Assume the number of objectives to be *nObj*. Before the computation of crowding distance in each iteration, the solutions have been sorted by the objective values in ascending order. To the first and last solutions, the distances are set as positive infinity. The crowding distance of solution i is calculated as

$$\mathcal{D}(i) = \sum_{m=1}^{m=nObj} \frac{Obj_m(i+1) - Obj_m(i-1)}{f_m^{max} - f_m^{min}}, \qquad 0 < i < |A| \tag{4}$$

$Obj_m(i+1)$ refers to the m^{th} objective of the $(i+1)^{th}$ solution. f_m^{max} and f_m^{min} are the maximum and minimum values of the m^{th} objective in the population. A solution with a smaller crowding distance means it is more crowded by it's neighboring solutions.

Using the first phase as an example, if only one solution's constraint violations are determined to be the worst and the value is more than the threshold, the solution is discarded, just like the original logic in IVEA [1]. However, if a set of solutions share the same number of constraint violations and have been determined to all be tied for the worst solutions, the one with least crowding distance value is discarded. Similarly, during the second phase, if multiple solutions have the same worst $\mathcal{U}$ value, the solution with maximum density, i.e., the least crowding distance, is eliminated. The environment selection procedure is listed in Algorithm 1.

Algorithm 1 EnvironmentSelection(S)

Input: S(population), n(population size)
Output: S(selected population with size n)

```
1: Select by the first-dimensional fitness: Iterate the following steps
   until the size of population $S$ does not exceed $n$ or the worst violation
   value does not exceed the threshold:
2: build a collection of solutions $\mathbf{V}$;
3: for each solution $x \in S$ do
4:     if $x$ has the largest violation value, i.e., $\mathcal{I}(x^*) < \mathcal{I}(x)$ for all $x^* \in S$
5:     $x \to \mathbf{V}$
6: end for
7: if $|\mathbf{V}| == 1$ then
8:     remove the solution in $\mathbf{V}$ from $S$;
9: else
10:    select the solution $x$ with the least crowding distance, i.e., $\mathcal{D}(x) <$
       $\mathcal{D}(x^*)$ for all $x^* \in V$ ;
11:    remove solution $x$ from $S$;
12: end if
13: Select by the second-dimensional fitness: Iterate the following steps
    until the size of population $S$ does not exceed $n$:
14: build a collection of solutions $\mathbf{V}$;
15: for each solution $x \in S$ do
16:    if $x$ has the largest $\mathcal{U}$ value, i.e., $\mathcal{U}(x^*) < \mathcal{U}(x)$ for all $x^* \in S$
17:    $x \to \mathbf{V}$
18: end for
19: if $|\mathbf{V}| == 1$ then
20:    remove the solution in $\mathbf{V}$ from $S$
21: else
22:    select the solution $x$ with the least crowding distance, i.e., $\mathcal{D}(x) <$
       $\mathcal{D}(x^*)$ for all $x^* \in V$;
23:    remove solution $x$ from $S$;
24: end if
```

3.4. Mutation

Mutation is an essential mechanism to improve the entire population by creating new genes. Sayyad et al. planted one *feature-rich* valid solution (a seed) in the initial population, and showed that the seed had critical influence on the generation of valid solutions, especially for large feature models [3]. Here, *feature-rich solution*, means a configuration in which "a sufficiently large number of features" have been selected. Inspired by this idea, we design a novel mutation operator to promote the scalability of IVEA-II.

Similarly with the seeds in Sayyad's work [3], a certain number of valid solutions are prepared beforehand. The mutation operator begins by selecting one solution from the population at random. It then further determines the procedure of changing the gene of this solution randomly. This occurs by either flipping one random bit of the gene or replacing one prepared valid solution with this solution. In order to improve the efficiency of the algorithm, the flipping operation will not be performed on core or dead features.

A vital question raised is how many valid solutions should be prepared for the replacements. Sayyad et al. certified that in the initial population one planted *feature-rich* valid solution efficiently outperform 30 random ones at producing valid solutions [3]. However, there are some differences between population initiation and mutation. In this work, we pre-compute 31 valid solutions to be used in the mutation for every feature model. These pre-computed values include one *feature-rich* solution and 30 randomly selected solutions. According to Sayyad et al.[3], three hours is required to generate one *feature-rich* seed with two-objectives. This time consuming process is essential because IBEA requires such a seed to minimize the rule violations and maximize the number of selected features. However, the common solutions can be obtained more easily. For the most complex *Linux* model, we obtained 286 valid solutions in 9.5 hours (about two minutes for one solution) through improving the Guided Improvement Algorithm (GIA) [17] by promoting the original linear search into a parallel non-dominated search. Since several approaches can be used to generate the valid solutions, such as several MOEAs, we do not list our improved GIA specifics here.

There are two main considerations during the determination of seeds number: a) During the evolutionary process, mutation is conducted far more often than initiation. In general, one run of an algorithm only has one initiation but hundreds of mutations. Multiple replicas of a single solution brought by mutation may have a limited impact on the generation of valid solutions; b) Besides the feature model conformance, the quality of optimal solutions is also critical. Intuitively, multiple replicas of one solution degrade the diversity of a population. Additionally, it is hard to promise the preservation of one valid solution during the evolution because of the balance on the multiple objectives.

Another important design aspect of IVEA-II is the collection of valid solutions. Differentiating from the traditional Pareto ranking as the last step in IBEA and SATIBEA, we continually collect valid solutions during the entire optimization process rather than only at the final step. Once a solution conforming to the feature model is created, the valid solution set is checked and the solution is added to the set if permitted. There are three rules to decide if the new valid solution will be collected. 1) The size of the valid solution set cannot exceed the population size set beforehand. 2) The solutions in the valid set must be unique. 3) Once the capacity of the valid set is reached, the dominated solutions are replaced by the non-dominated ones. The collection process is described as Algorithm 2.

Algorithm 2 CollectValidSolution($solution_i$,S_v**)**

Input: $solution_i$ (a new valid solution), S_v(valid solution set), n(population size)
Output: valid solution set S_v

```
1:  if $|S_v| < n$ then
2:      if $solution_i \notin S_v$ then
3:          $solution_i \rightarrow S_v$
4:      end if
5:  else
6:      for each $solution_j \in S_v$ do
7:          if $solution_i \succeq solution_j$ then
8:              $S_v = S_v \setminus \{solution_j\}$
9:              $solution_i \rightarrow S_v$
10:         end if
11:     end for
12: end if
```

4. Research Questions

Recall that the final goal of this work is to evaluate the scalability of our proposed IVEA-II. This goal can be interpreted as whether IVEA-II can generate more valid solutions with higher quality in a fixed amount of time on large-scale feature models. The goal can be further decomposed as the following three research questions.

RQ1: How efficient is IVEA-II on generating valid solutions?

This question is concerned about the number of unique valid solutions generated by IVEA-II and the baselines over a fixed time period. This number also illustrates the diversity of approximate solutions and decides how many feasible and optimal configurations a user can choose. Here, we only care about the scale of valid solutions, because only these valid solutions make sense in practice.

This question has two implications. Firstly, we focus on a single algorithm. How will the efficiency change with the scales of feature models? Will the number of valid solutions decrease sharply with the increased scale of feature models? Secondly, we explore the distinctions between IVEA-II and the other state of art algorithms. Using these two questions, we believe the efficiency can be fully evaluated.

While RQ1 focuses on the efficiency of valid solution generation, we also must be concerned about the quality of these valid solutions.

RQ2: Do the approximate solutions generated by IVEA-II have better quality than those created by the current state of the art algorithms?

Selected solutions not only need to conform to the feature model, but also have optimized multiple objectives. Thus, after RQ1, we evaluate the quality of the valid solutions made by IVEA-II and the other leading algorithms.

It is common knowledge that convergence and diversity are two main goals in multi-objective optimization [5]. Therefore, we would like to explore the quality of all valid solutions and compare the different approaches from these two aspects.

Given that IVEA-II performs better on both the generation efficiency (RQ1), and the quality of valid solutions (RQ2), we then want to investigate the inner designs of IVEA-II, leading us to the third research question.

RQ3: To what extent does each facet of the design in IVEA-II impact the feature selection? Are all of the design pieces necessary for all feature models?

As demonstrated above, there are three important points in IVEA-II: the 2D fitness function, crowding distance and the new mutation operator. It is safe to assume that the 2D fitness function may increase the number of valid solutions but also destroy diversity as well. This begs the question of what impact will our 2D fitness function have on the valid and optimal solution generation? Crowding distance is a well-known diversity preserving mechanism, but does it really have positive effects on the diversity of the final approximate solutions for all feature models? If so, how much?

In our new mutation operator, 31 valid solutions (i.e., one *feature-rich* and 30 random selected solutions) worked as seeds are prepared beforehand. The mutation will pick one solution at random and replace it with a solution in the population with a certain probability. In light of the experimental results previously [3], the valid solutions will most likely improve the search process. Even more, Sayyad et al. observed that using one *feature-rich* seed is enough for improving the optimized process and even can obtain better performance than using 30 randomly selected ones [3]. Is this conclusion suitable for the mutation operator, or should we use one carefully selected seed instead of adding more common ones?

5. Experimental Setup

In order to answer the research questions, a series of experiments have been conducted in this section. This section firstly describes experimental settings, including the feature models, optimization objectives and the parameter settings of IVEA-II, SATIBEA and IBEA. Then, some metrics used to measure the performance of these algorithms are explained.

5.1. Settings

5.1.1. Feature Models and the Optimization Objectives

To evaluate whether our technique can run on large feature models effectively, we select five large models from the Linux Variability Analysis Tools (LVAT) repository [1]. These five models were also adopted in two previous papers [15, 3] and we believe they are enough to evaluate the scalability. The characteristics of these five feature models are listed in Table 1.

Following the previous studies [2, 3, 15], each feature is augmented with three attributes. These three attributes, *Cost*, *Used-before* and *Defects* are assigned arbitrarily. *Cost* takes real values distributed normally between 5.0 and 10.0. *Used-before* indicates whether a feature has been used or not, using a uniformly distributed Boolean value. *Defects* counts the number of

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

Table 1: The Settings of Feature Models

FM[1]	**Version**	**Features**	**CF**[2]	**DF**[3]	**Constraints**	**Reference**
eCos	3.0	1244	0	35	3146	She et al.[18]
FreeBSD	8.0.0	1396	3	38	62183	She et al.[18]
Fiasco	2011081207	1638	49	964	5228	Berger et al.[19]
uClinux	20100825	1850	7	1237	2468	Berger et al.[19]
Linux	2.6.28.6	6888	58	102	343944	She et al.[18]

[1] FM: Feature Models
[2] CF: Core Features
[3] DF: Dead Features

errors that occur during the implementation of a feature, using integer values distributed normally between 0 and 10.0.

Based on these three attributes, five goals are set to be optimized:

i **Correctness:** the selected features have to comply with all of the constraints of a feature model.

ii **Reuse:** maximize the number of features that have been used before.

iii **Less error:** minimize the total number of defects caused by all selected features.

iv **Lower cost:** minimize the sum of the cost of all selected features.

v **More features:** maximize the number of selected features.

5.1.2. Algorithm Settings and Implementations

We follow the algorithm settings set in place previously [15], listed in Table 2. We would like to emphasize an important point: the number of *runs*. Due to some operations in the algorithms introducing randomness, such as crossover and mutation, we should run the algorithms multiple times to reduce the effect of stochastic behaviors on the comparisons. The question is then: how many runs are appropriate? According to Arcuri et al.[20], 30 observations is a common rule of thumb in many fields. Additionally, considering the number of feature models and the relatively large scales of feature models in our experiments, we independently run each algorithm 30 times.

Table 2: The Parameter Settings of Algorithms

Parameter	Setting	Parameter	Setting
population size	300	max evaluations	25000
crossover probability	0.05	crossover type	single point
mutation probability	0.001	independent run	30

The three algorithms in this paper were implemented on the basis of jMetal[2] [16]. We would like to thank Henard et al. [15] for making the source code of SATIBEA and the related intermediate data available publicly because this provided convenience for our comparison experiments. Due to the clear structure and sound readability, we refactored the code of IVEA-II and implemented it based on the framework of SATIBEA. IVEA-II can be accessed in *https://github.com/tomato12/FeatureSelection*.

All the algorithms in our experiments were run on a server with an Intel(R) Xeon(R) E5-2420 1.90GHz 6 core CPU and 24.0 GB RAM.

5.2. Metrics

There are two common goals in multi-objective optimization [5]: 1) Convergence, which is the closeness to the Pareto Fronts; 2) Diversity, which is the wideness distributed among the Pareto Fronts.

These two goals cannot be measured by a single performance metric. After a detailed investigation [15, 2, 3, 21], we adopted four commonly used metrics. The metrics are *Hypervolumn*, *Epsilon*, *Inverted Generational Distance* and *Spread*. We also regard the number of valid optimal solutions as another metric to assess the diversity and effectiveness of an algorithm.

1) **Convergence Metrics**

(a) **Epsilon(ϵ)**, which is defined as the factor by which a solution set is worse than another with respect to all objectives [22]. More preciously, the definition of *Epsilon* is as follows [22]: Given that a smaller objective value is preferable, an objective vector $z^1 = ({z_1}^1, {z_2}^1..., {z_n}^1)$ is said to ϵ-dominate another objective vector $z^2 = ({z_1}^2, {z_2}^2..., {z_n}^2)$, if and only if

$$\forall 1 \leq i \leq n : {z_i}^1 \leq \epsilon \cdot {z_i}^2 \tag{5}$$

[2]http://jmetal.sourceforge.net/

for a given $\epsilon > 0$. The ϵ-indicator I_ϵ is defined as

$$I_\epsilon(A, B) = \underset{\epsilon \in \mathbb{R}}{inf}\{\forall z^2 \in B, \exists z^1 \in A : z^1 \succeq_\epsilon z^2\} \tag{6}$$

A lower ϵ means a closer distance from one solution set to another.

(b) **Inverted Generational Distance (IGD)** is defined as the measure of quality in a solution using a reference set [23]. Let $P = \{P_1, P_2, ..., P_{|P|}\}$ be the reference set where $|P|$ is the cardinality of P. The approximate solution set is A. Then the *IGD* between A and P can be defined as follows.

$$IGD(A, P) = \frac{\sum_{\tau \in P} d(\tau, A)}{|P|} \tag{7}$$

$d(\tau, A)$ is the Euclidean distance from $\tau \in P$ to its nearest objective in A. A lower *IGD* value means a closer distance from A to P.

2) **Diversity Metrics**

a) **Spread** is defined as the measure of "the extent of spread achieved among the obtained solutions" [5]. The more formal definition can be seen from the bellow formula. Here, d_i is the Euclidean distance between consecutive solutions in the obtained non-dominated set of solutions, and $\overline{d}$ is the average of all these distances. d_f and d_l are the Euclidean distances between the extreme solutions and the boundary solutions. Assuming there are N optimal solutions, *(N-1)* consecutive distances are made. A smaller Δ means more similar consecutive distances. Specifically, to a uniform distribution, the Δ is zero.

However, in this paper, we would prefer a larger Spread, as in the studies [2, 15]. The higher *Spread* value reflects a better diversity of the solutions, with different Euclidean distances from the average distance.

$$\Delta = \frac{d_f + d_l + sum_{i=1}^{N-1}|d_i - \overline{d}|}{d_f + d_l + (N-1)\overline{d}} \tag{8}$$

b) **Hypervolume (HV)** is the measure of the volume of the objective space that is dominated by the solution set [24]. We use the description in Zitzler et al. to give the explanation about *Hypervolume* [22]. The closed volume is formed by a reference set and the Pareto Front. Let the Pareto Front be $RF \in \mathbb{R}$ and the reference set be $\mathbf{r} = (r_1, r_2..., r_c) \in \mathbb{R}^{\mathbf{c}}$. Hypervolume indicator HV can be formalized as

$$HV(PF, \mathbf{r}) = \lambda(\underset{s \in PF}{\cup} space(\mathbf{s}, \mathbf{r})) \tag{9}$$

with $space(\mathbf{s}, \mathbf{r}) = \{\mathbf{v} \in \mathbb{R} | \mathbf{r} \prec \mathbf{v} \preceq \mathbf{s}\}$ being the hyperplane. The hyperplane contains all vectors $\mathbf{v}$ that are weakly dominated by $\mathbf{s} \in PF$ and dominate $\mathbf{r}$. λ is usually the Lebesgue measure [25]. A higher *Hypervolume* value is preferred.

6. Experimental Results

6.1. Run Time for Valid Solutions

RQ1 asks about the efficiency of generating valid solutions. We answer this question from two angles. How many valid solutions are generated during a fixed time period and how long does a valid solution take?

To start with, we investigate the number of valid solutions generated by the three algorithms. Just as the discussion above, only the solutions conforming to all constraints of feature models make sense in practice. Thus, the number of valid solutions a feature selection approach can make is an important metric in evaluation. We would like to concentrate on the algorithm producing the most valid solutions, but at the same time, which factor of a feature model has the biggest impact on the number of solutions for a single algorithm, the feature number or the scale of constraints? The answers can be found in Figure 3. The difference between the two sub-graphs (a) and (b) is the order of feature models in the horizontal axis. The left sub-graph (a) arranges the feature models by sort ascending according to the number of contained features, while the right (b) orders constraints from least to greatest in a feature model.

Looking at Figure 3, it is fairly straightforward to observe that IVEA-II always produces more valid solutions than the other two algorithms for all feature models. Specifically, 9,000 (i.e., 300 population size times 30 runs) valid solutions were captured for all five feature models. At the same time, SATIBEA outperforms the improved IBEA.

Naturally, more valid answers would be generated for simpler feature models and researchers often use the feature number as the most important metric to evaluate the scale of a model. However, we can observe clearly that more feasible solutions of *uClinux* were generated than that of *eCos*, *FreeBSD* and *Fiasco*, all of which have less features than *uClinux*. Here, recall that *uClinux* has less constraints than these three feature models shown in Table 1. Figure 3 tells us that the constraint number has a more critical role on the amount of valid solutions generated for all of the three algorithms. From this analysis, we can make our first observation.

Observation 1: The amount of constraints in a feature model has more influence on the efficiency of an algorithm to produce valid solutions than the number of features does.

The three algorithms used took place during a fixed time period. Therefore, more valid solutions generated mean less time spent on an individual solution. To show this clearly, we list the average time for one valid solution in Table 3. For every algorithm, 30 runs took 90 minutes where three minutes was set for one run. No valid solutions were generated by IBEA for *FreeBSD*, so we marked the corresponding cell with a '-' symbol. It can be observed that our IVEA-II can make one valid solution for all the five feature models every 0.6 seconds by average.

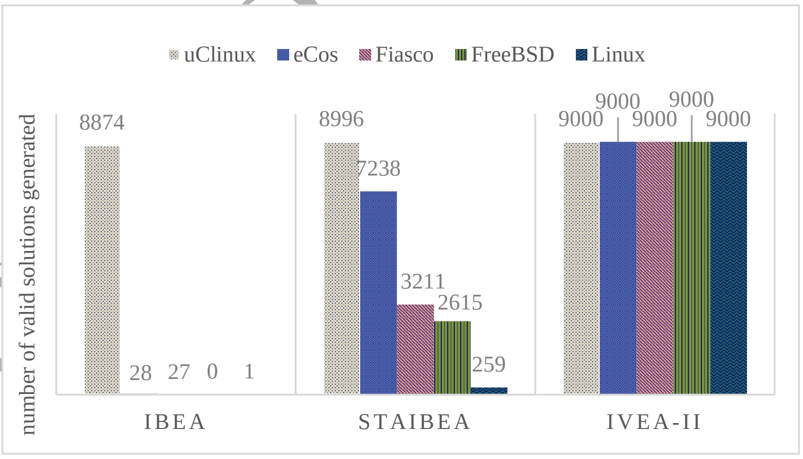


Figure 3: Number of Valid Solutions generated by each of the three algorithms: the feature models are listed by the number of constraints in descending order.

Table 3: Average Elapsed Time for One Valid Solution

	ecos (secs)	**FreeBSD** (secs)	**Fiasco** (secs)	**uClinux** (secs)	**Linux** (secs)
IBEA	19.29	-	200	0.61	5400
SATIBEA	0.75	2.06	1.68	0.60	20.85
IVEA-II	**0.60**	**0.60**	**0.60**	**0.60**	**0.60**

One aspect not listed is the "no meaningful" valid solutions, which do not contain any features but keep all rules of a feature model, named as "zero-feature" solutions previously [3]. This occurs on the feature models without core features, just like the model *eCos*. We noticed that SATIBEA accepted 621 such "zero-feature" solutions for *ecos* in all, or rather 8.6% of all valid solutions created by SATIBEA. By contrast, no such solutions were created by our IVEA-II, showing an obvious disadvantage of SATIBEA.

Because of the noticeable gap between IBEA and the other two algorithms, i.e., SATIBEA and our IVEA-II, and the outperformance of SATIBEA against IBEA shown in the past with the same data [15], we would not replicate the work of [15]. In the following sections, we only analyze the performance of SATIBEA and IVEA-II.

6.2. Quality of Valid Solutions

This section explores the second of our research questions involving the quality of optimal solutions generated by IVEA-II and STAIBEA. We investigate the answers by comparing the optimization objectives and quality indicators mentioned in Section 2.

The evaluations of optimization objectives are reported in Table 4. The upper table area *Comparing the Original Objectives* shows the evaluations on the direct objectives. As the illustrations in Section 5.1 show, there are five objectives to be optimized in the present paper: conforming to the feature model, minimizing the number of deselected features (*unSelected* in Table 4), minimizing the number of features that were not used before (*unUsed* in Table 4), minimizing the defects caused by the selected features (*defects* in Table 4) and minimizing the cost of all selected features (*cost* in Table 4). Due to the analysis being conducted on only the valid solutions, the first objective "correctness" is not listed here.

Indicators of *HV*, *SPREAD*, *EPSILON* and *IGD* were used to measure the convergence and diversity of these optimal solutions, listed under the section entitled *Comparing the Quality Indicators on the Optimization Objectives* in Table 4. Since all algorithms were run 30 times, we therefore can get at most 30 values for each indicator.

To deal with the impact of randomness within IVEA-II and SATIBEA, we used a statistical test to provide support for data analysis, suggested by Arcuri et al. [20]. The Mann-Whitney Wilcoxon-test (U-test) is a nonparametric test, which can be applied on an unknown distribution, instead of only on a normal distribution like the *t-test*. We used a two-tailed U-test at a significance level of 0.05 to get a p-value of the lowest significance level for which the null-hypothesis would not be rejected. p-value < 0.05 indicates that the two algorithms have significant differences with moderately strong evidence. In our context, it means that there are significant differences between the results of IVEA-II and its counterpart. A p-value > 0.05 means there is very weak evidence that the null hypothesis does not hold. Since multiple objectives of the same population are compared, one vital aspect we have to consider is the increased probability of incorrectly rejecting the null hypothesis. The Bonferroni correction is a common way used to counteract this problem [26]. Using the Bonferroni correction, if m hypotheses are tested in an experiment, the statistical significance level of each individual hypothesis is adjusted as 1/m of the desired maximum overall level. Here, four comparisons are performed on each pair of the population. The individual significance level is therefore adjusted as 0.0125 (i.e.,0.05/4). If the p-value is greater than 0.0125 in a comparison, we can say no significant difference can be found from the two algorithms. In Table 4, we highlighted these cells with gray color.

Using the p-value to indicate if there exists a significant difference between the two approaches, we then want to know which algorithm is better. We utilized $\hat{A}_{12}$ statistics, which measures the probability that algorithm $\mathcal{A}$ yields higher values than algorithm $\mathcal{B}$ [20, 27]. In our context, not all performance measures (i.e.,indicators and objectives) with higher values are preferable, we thus slightly modified the equation of $\hat{A}_{12}$ by calculating the probability of better values obtained by algorithm $\mathcal{A}$ than that of algorithm $\mathcal{B}$ to get a uniform criteria for judging diverse measures. If $\hat{A}_{12}$ is equal to 0.5, the two algorithms are equivalent. If $\hat{A}_{12}$ is above 0.5, algorithm $\mathcal{A}$ works better than algorithm $\mathcal{B}$.

In both of the objective and indicator comparisons within Table 4, we firstly count the comparisons with no significant difference (p-value with gray color). Then, we count the times IVEA-II performs better (regular

p-value and bold $\hat{A}_{12}$). Lastly, we count the times that IVEA-II performs worse (regular p-value and regular $\hat{A}_{12}$). We can observe that in the 20 comparisons on the original objectives, six have no significant difference. In the next 14 comparisons, IVEA-II wins 11 times. This result illustrates that our approach IVEA-II obtained better objective values for the five feature models on the five objectives from a statistical viewpoint. The comparisons on the convergence and diversity of these optimal solutions can be seen on the lower portion of the table. It is clear that within the 20 comparisons, four have no statistical difference and of the remaining 16 comparisons, IVEA-II wins 11 times, and loses five times.

These results illustrate that IVEA-II outperforms SATIBEA in both the objective values and on the quality of the optimal solutions.

Table 4: Comparing IVEA-II against SATIBEA: the Wilcoxon Test on the original multiple objectives and quality indicators at the significance level of 0.0125 adjusted by Bonferroni Correction

	Comparing the Original Objectives							
	unSelected		**unUsed**		**defects**		**cost**	
	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$
ecos	<2.2e-16	**0.64**	<2.2e-16	0.36	<2.2e-16	**0.83**	<2.2e-16	**0.96**
FreeBSD	<2.2e-16	**0.62**	<2.2e-16	0.38	<2.2e-16	**0.76**	<2.2e-16	**0.91**
Fiasco	1.93e-02	**0.51**	5.07e-02	0.49	6.13e-03	**1**	3.63e-03	**1**
uClinux	<2.2e-16	**0.54**	<2.2e-16	0.46	5.10e-08	**0.80**	<2.2e-16	**0.92**
Linux	3.7e-01	0.48	3.3e-01	**0.52**	4.7e-01	**0.81**	3.3e-01	**0.87**
result	In 20 comparisons, 6 find no significant difference; IVEA-II wins 11 times, loses 3 times.							

	Comparing the Quality Indicators on the Optimization Objectives							
	HV		**SPREAD**		**EPSILON**		**IGD**	
	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$
ecos	<2.2e-16	0	<2.2e-16	**1**	0.10	0	0.06	0.62
FreeBSD	4.0e-03	0.29	7.24e-10	**0.92**	5.44e-03	**0.71**	5.07e-16	**0.99**
Fiasco	3.43e-01	0.43	<2.2e-16	**1**	<2.2e-16	**0.88**	<2.2e-16	**1**
uClinux	<2.2e-16	0	<2.2e-16	**1**	2.65e-11	0	3.0e-03	0.28
Linux	0.83	0.52	<2.2e-16	**1**	4.42e-11	**1**	<2.2e-16	**1**
result	In 20 comparisons, 4 find no significant difference; IVEA-II wins 11 times, loses 5 times.							

6.3. Evaluation of Each IVEA-II Feature

From Section 6.2, we can see that IVEA-II outperforms SATIBEA. IVEA-II is composed of three critical features. In our third research question, we were concerned about which features work and if all features are necessary across all feature models. To find an answer, we designed four groups of experiments to evaluate each feature. The optimal solutions generated by IVEA-II, with all three features, is used as the "golden-answer". The four contrasting algorithms, reserving all of the features except for the target one to be evaluated, were implemented respectively. The method of comparing results in Section 6.2 was also used in these experiments and the results are listed in Table 5. Additionally, the number of valid solutions is also an important factor we must consider. Due to the competing and conflicting multiple objectives, no single solution is best. Users have to do their final determination from a series of non-dominated solutions according to some extra information. Enough choice space is therefore critical for users. We recorded the number of valid solutions of the largest and most complex model *Linux* created by different algorithm settings listed in Figure 4, for the consideration of scalability. Another reason is that IVEA-II with all of the different settings can generate the maximal (i.e., 9000) valid solutions for the other four feature models.

The first feature we examine is the **2D fitness function**. Although it may be helpful for valid solution generation, it is also possible to be a detriment on the quality of solutions. If this is true, the feature may not be worth it's benefit. The first experiment was to compare the results generated by IVEA-II with the 2D fitness function against IVEA-II using a normal fitness function (with/without 2D fitness in Table 5). The counterpart adopted the second dimension of the 2D fitness function, which is IBEA's original fitness function used in Sayyad's improved IBEA and SATIBEA. The first dimension is also transferred into one extra optimization objective just like in previous studies [3, 15]. We observed that in 20 comparisons, 10 found no significant difference. In the remaining 10 comparisons, the version of IVEA-II with a 2D fitness function wins five times. This shows the 2D fitness function almost has no negative impact on the quality of solutions, as expected. From Figure 4, we can see that only 5,064 valid solutions are created by the algorithm without a 2D fitness function and 722 additional valid configurations are created with the function. This presents an important observation.

Observation 2: The reduction of search space does not necessarily impair the quality of the generated optimal solutions, just like our 2D fitness function.

Table 5: Comparing different settings of IVEA-II: the Wilcoxon Test on the indicators at the significance level of 0.05. The significance level on each individual comparison is adjusted as 0.0125 by Bonferroni Correction. (with/without 2D fitness; with/without CD; our mutation with 31 seeds vs. common bit flip mutation, our mutation with 30 common seeds vs. 1 feature-rich seed)

Pair of Alg.	**FM**	**HV**		**SPREAD**		**EPSILON**		**IGD**	
		p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$
with/ without 2D fitness	eCos	5.32e-02	0.35	4.49e-01	0.44	5.18e-06	0.16	1.0e-02	0.31
	FreeBSD	1.46e-01	0.39	3.19e-02	**0.66**	9.31e-02	**0.63**	3.90e-01	0.43
	Fiasco	1.98e-13	0.03	3.85e-02	**0.66**	9.98e-08	**0.89**	9.20e-04	0.26
	uClinux	1.87e-01	0.40	4.42e-01	**0.56**	2.62e-03	**0.72**	9.82e-01	0.50
	Linux	2.35e-15	0.01	4.38e-12	**0.95**	1.81e-09	**0.95**	<2.2e-16	**0.99**
result	In 20 comparisons, 10 find no significant differences;the settings with 2D fitness wins 5 times								
with/ without CD	eCos	1.59e-01	0.39	8.78e-01	**0.51**	7.62e-07	0.13	9.21e-04	0.26
	FreeBSD	6.13e-01	**0.54**	1.64e-01	**0.61**	5.69e-01	**0.54**	5.33e-01	**0.55**
	Fiasco	9.65e-02	**0.63**	2.24e-02	0.33	1.03e-01	0.38	3.66e-01	**0.57**
	uClinux	6.12e-01	**0.54**	7.23e-01	**0.53**	5.88e-01	**0.54**	9.12e-01	**0.51**
	Linux	<2.20e-16	0.01	4.38e-12	**0.95**	1.42e-08	**0.92**	<2.2e-16	**1.0**
result	In 20 comparisons, 14 find no significant difference; the settings with CD wins 3 times								
with/ without our mutation	eCos	3.21e-16	**0.99**	<2.20e-16	**1.0**	5.60e-03	**0.71**	5.13e-01	0.55
	FreeBSD	4.23e-01	0.44	<2.20e-16	**1.0**	2.43e-03	**0.73**	3.26e-05	**0.80**
	Fiasco	<2.20e-16	**1.0**	<2.20e-16	**1.0**	3.08e-11	**1.0**	9.65e-02	**0.37**
	uClinux	<2.20e-16	**1.0**	<2.20e-16	**1.0**	8.12e-10	**0.96**	1.94e-08	0.11
	Linux	<2.20e-16	**0.53**	<2.20e-16	**1.0**	2.99e-11	**1.0**	2.20e-16	**1.0**
result	In 20 comparisons,3 find no significant difference;the settings with our mutation wins 16 times								
30/1 seeds in mutation	eCos	<2.20e-16	**1.0**	<2.20e-16	0	1.60e-01	**0.61**	1.87e-01	**0.60**
	FreeBSD	1.06e-01	**0.62**	3.07e-08	**0.89**	5.44e-01	**0.45**	1.91e-02	**0.68**
	Fiasco	<2.20e-16	**1.0**	5.27e-12	0.05	4.59e-10	**0.97**	4.11e-06	**0.83**
	uClinux	<2.20e-16	**1.0**	<2.20e-16	0.01	3.94e-09	**0.94**	4.30e-02	0.35
	Linux	1.08e-08	**0.90**	2.60e-03	**0.72**	9.52e-04	0.25	1.29e-03	**0.74**
result	In 20 comparisons,6 find no significant difference; the mutation with 30 seeds wins 10 times								

Next, we analyze the impact of the **crowding distance (CD)** on optimal feature selection. From the work of Deb et al.[5], we know that CD

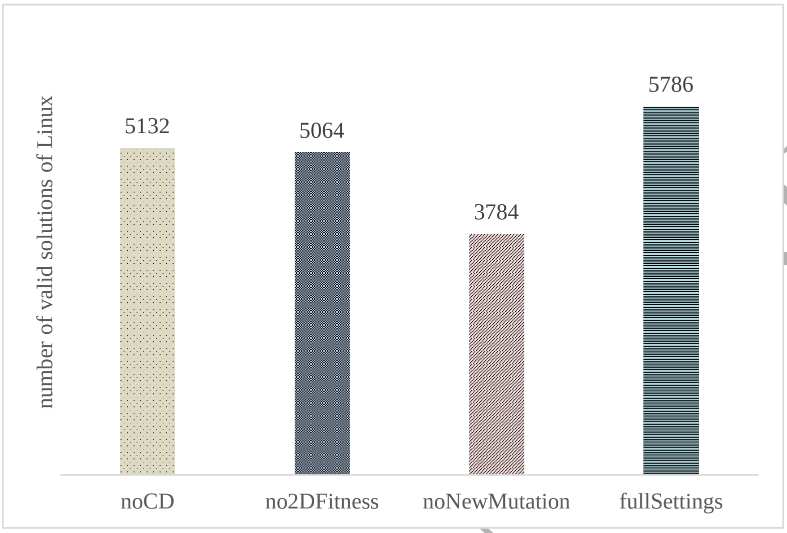


Figure 4: The number of valid solutions of *Linux* generated by IVEA-II with different settings.

Table 6: Number of features in all seeds

FM	**feature number in the feature-rich seed**	**feature numbers in the 30 randomly selected seeds**
eCos	1148	560, 648, 943, 681, 1092, 875, 681, 945, 648, 1127, 1080, 594, 164, 286, 877, 649, 231, 533, 178, 427, 26, 281, 568, 335, 330, 333, 163, 889, 431, 873
FreeBSD	1088	423, 529, 1055, 311, 102, 1056, 756, 1080, 64, 281, 281, 758, 764, 1135, 480, 422, 535, 1136, 284, 758, 310, 532, 1077, 419, 480, 540, 586, 529, 753, 422
Fiasco	358	332, 236, 192, 229, 186, 285, 226, 180, 202, 270, 284, 233, 205, 302, 339, 180, 316, 351, 320, 261, 312, 249, 246, 261, 249, 303, 213, 331, 345, 203
uClinux	613	7,477, 525, 129, 81, 311, 191, 207, 355, 287, 515, 351, 59, 13, 219, 267, 175, 127, 397, 193, 613, 59, 417, 27, 335, 441, 265, 159, 149, 109
Linux	6232	3427, 3401, 3491, 208, 3315, 3401, 3501, 3453, 3390, 3387, 3391, 6224, 2892, 193, 3272, 6220, 5296, 199, 181, 186, 186, 151, 6161, 2866, 212, 3261, 2910, 141,178, 3314

is a brilliant mechanism to improve the diversity of optimal solutions. Does it work within our context? We performed an experiment to compare the quality of optimal solutions generated by IVEA-II with CD against the algorithm without CD (with/without CD in Table 5). The results show that in 20 comparisons using these two settings, no significant difference can be found in 14 of the comparisons. In the remaining six, the version of IVEA-II using CD wins three times, equal to that of the counterpart. Therefore, CD does not present the same distinguished effect on the quality of the final approximate solutions as our expectation. One possible reason is that we adopt another diversity control mechanism, which is the binary indicator I_{HD} in the second dimension of our 2D fitness function, borrowed from IBEA. This fitness function emphasizes users' preference on the quality of the approximate solutions, and its significance on optimized feature selection has been proven by existing research [2]. We will do further investigations in future. However, from Figure 4, we notice that the algorithm with CD can make 654 additional valid solutions (i.e., 5,786 - 5,132) for *Linux* than the algorithm without CD. We believe it is reasonable to conclude the following observation.

Observation 3: Crowding distance is beneficial for the generation of unique valid solutions, even though it may have no profound effect on the quality of the optimal solutions.

Finally, we focus on our **mutation operator**. Intuitively, planting sound solutions with distinguished genes into the population can impact the evolutionary process and enhance the whole population's quality. Focusing on this idea, we prepared 31 valid solutions as seeds, consisting of one *feature-rich* seed and 30 randomly selected fully-correct solutions, for mutation. Then, we replace one of these 31 seeds with an individual selected randomly from the population with a certain probability. The probability can be tuned in concrete applications. The selection of one seed is also stochastic. The feature numbers in these 31 seeds are shown in Table 6.

One question that arises is, to what extent does this new mutation impact the quality of the optimal valid solutions? Another question is whether the 31 seeds used in the mutation are appropriate if one seed with "high quality" is enough, like that in initialization operation suggested in another study [3]? We designed two groups of experiments to look for the answers.

We began by attempting to evaluate the new mutation with all 31 seeds. IVEA-II is ran with two different mutation operators: our mutation and the common *BitFlipMutation* in *jMetal*. Other than the mutation, the remaining operators are the default settings in IVEA-II. The detailed results can be seen in the table area *with/without our mutation* in Table 5. The settings with our mutation obtain remarkable performance. In 20 comparisons, only three have no significant difference. In the remaining 17 comparisons, the settings with our mutation wins 16 times. From Figure 4, it is observed that our mutation has a critical influence on the number of valid solutions. To be more specific, 2,002 additional valid solutions of model *Linux* are generated with our mutation.

We did further evaluation to explore the influence of seed number on the optimization selection. Due to suggestions about the number of seeds [3], we compare the settings with 30 randomly selected seeds in our mutation against one *feature-rich* seed. The related results can be seen in the rows *30/1 seeds in mutation* of Table 5. In 20 comparisons, six have no significant difference. In the other 14 comparisons, the settings with 30 seeds in mutation wins 10 times.

In addition, we recorded the elapsed time for creating the first 300 valid solutions of three models generated by these two different settings in Figure 5. These three models (i.e., *FreeBSD*, *eCos* and *uClinux*) were selected because their elapsed time for the valid solution generation is relatively close so that the visualization of this diagram is more clear. From it, we can see that the settings with 30 common seeds in mutation can create valid solutions faster than that with one *feature-rich* seed in the mutation. This conclusion contrasts points made by Sayyad et al. about the initialization [3]. One potential reason is that the initialization and mutation are different both on the performed times during the whole evolution process and the impact of creating new offspring.

Thus, we have two more observations.

Observation 4: The valid solutions put in the population through mutation can help the creation of valid solutions. In fact, it notably enhances the quality of the valid optimal solutions.

Observation 5: Multiple good enough seeds (i.e., 30 randomly selected seeds) in mutation process have more influence upon both the number and quality of the generated valid solutions than one best seed (i.e., the *feature-rich* one).

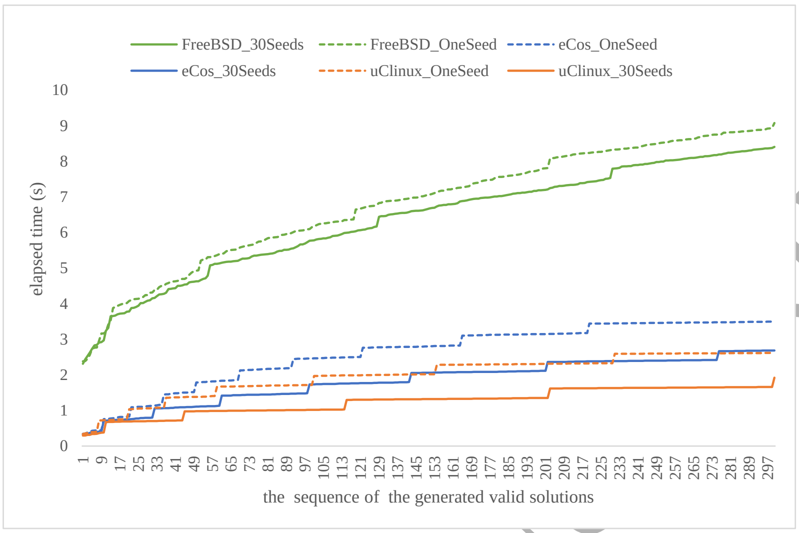


Figure 5: The elapsed time of generating valid solutions of three models by the settings with a different number of seeds in mutation

7. Discussion

7.1. Practical Implications

The optimization objectives in this work are general, and we do not limit the concrete types. As an extension, IVEA-II can also deal with the optimization of multiple non-functional requirements (NFRs) as demonstrated in [1]. Here the NFRs can be qualitative or quantitative. The NFRs can also have some numeric constraints or have none at all. The numeric constraints on NFRs, for example, cost $< \$1,000$, can be used as filters to discard the solutions with cost above \$1,000. During the selection process, every solution can be evaluated to decide if this inequality is violated. If violated, the solution is marked indicating it is invalid. We suggest that this invalid solution should continue attending the upcoming selections. Before returning the final results to users, all solutions with an invalid marking should be discarded. This way can avoid limiting search-space during optimization.

IVEA-II can also handle NFRs without numeric constraints. The NFRs can be qualitative or quantitative, but as long as they can be mapped into concrete objectives, our method works.

IVEA-II can also deal with the functional requirements (FRs) which are implemented by specific features. Therefore, FRs can be presented by some features which have to be selected in any solution, just like the core features. Unfortunately, the requirements, whether NFRs or FRs, add extra constraints to the selection which may make no feasible solution. We therefore recommend to evaluate whether valid solutions exist before the optimization process.

7.2. Threats to Validity

Threats to validity are discussed from internal, construct, external and conclusion validity.

Internal validity refers to how well the experiment is done so that a causal relationship can be concluded from the study. The biggest threat to internal validity is introduced by the use of synthetic attributes. Due to the lack of public data in real projects, we used synthetic attributes assigned by controlled distributions in experiments. In practice, the attributes will likely differ from the values we used. Different source data may lead to different results. In order to mitigate the bias, we adopted the same data across all three algorithms. Even more so, the data is identical to the data used in Henard et al. and other studies [3, 15], due to the source data being open access. Therefore, we believe it is feasible to say that our results are fair.

One additional threat to internal validity is from the evaluation of the number of seeds for our mutation operator. The effect of seed number on the efficiency of valid solution generation was evaluated by preparing one *feature-rich* and 30 common seeds for the mutation, and comparing the elapsed time for generating the same number of valid solutions. The quality of these seeds is therefore critical. In Sayyad et al. the quality means the number of features selected in the seed [3]. However, whether the selected features are just involved in constraints of feature models may be an important but uncertain factor for the valid solution generation. In order to alleviate this impact, we did the experiments on three different models.

Construct validity discusses the extent to which the goal that is designed to measure is accurately measured. When comparing the qualities of the valid optimal solutions generated by different algorithms, there is no single metric that can measure the *diversity* and *convergence* of the approximate solutions adequately. In order to compare the solutions in a comprehensive way, we adopt four different metrics *HV*, *SPREAD*, *EPSILON* and *IGD* to do the evaluation together. In each comparison, these four metrics were treated equally.

External validity is about the generalizability of our approach. In our experiments we applied our approach to five different feature models with diverse scales and complexities. Our approach doesn't limit some specific feature models, as long as the related attributes data and pre-computed seeds are available. On the other hand, the quality of seeds (i.e., the number of selected features in each seed) may impact the efficiency of generating valid solutions and the quality of the final approximate solutions on multi-objective optimization. We cannot guarantee that every reader can produce the same seeds we did since there are several possible approaches to generate valid seeds and we don't limit the process to a specific one. However, all of the seeds except the *feature-rich* one were selected randomly to avoid a bias in seed selection. To compare with the baselines and replicate their results, we used the same *feature-rich* seed with them. We also made our seeds public at the following address: *https://github.com/tomato12/FeatureSelection.*

Conclusion validity is the extent to which the relationship we reach is reasonable. In our work the major threat to conclusion validity is from the randomness of the applied algorithms, both on the initial population generation and the genetic operations, including crossover and mutation. These random activities could theoretically lead to the results of this paper by chance. For the initial population generation, we compared the five objective values of the initial population of two feature models in our previous work [1]. The results illustrated that no significant difference exists between the objectives of the initial population generated in the same way. Although the case models in our prior paper were smaller [1], the conclusion can also dispel the notion that the performance of IVEA-II is impacted by the difference on initial population. The reason for this is that the same approach is adopted to generate the initial population for the five different models besides the fixed assignments on the core and dead features (i.e. the bits of core features are set as 1, and dead features as 0). Whereas for the stochastic process in evolution, we addressed this problem by running each algorithm 30 times independently, and using the Wilcoxon test to compare the performance of IVEA-II and its counterpart. By using this statistical method, we largely improve the confidence of our analysis. Besides, we used Bonferroni Correction to adjust the significance level to decrease the probability of incorrectly rejecting the null hypothesis, due to the multiple statistical tests on the same data set.

7.3. Limitations

The first limitation we want to discuss is about the synthesis data usage in this work. Although it is reasonable to evaluate the three algorithms with the same synthesis data and conclude that our experimental results are fair, we can only say that our approach is practical significance after an evaluation with the attributes data of features and the optimized goals collected from real projects. There are some works about building and maintaining the traceability between features and source code in software product lines [28, 29, 30] which will be a prerequisite of collecting the attributes related with implementation (e.g. *footprint*) or run-time performance (e.g. *defects*). Some other great researchers discussed the impact of features to the quality performance of product variants [31], which is beneficial to selecting the features that are optimal for the specific quality.

The other limitation is that we only focus on *Boolean Variability Model*, in which all of the constraints can be transformed into a Boolean expression. The task of feature selection is essentially a process of deciding whether a feature should be selected or excluded in our present paper and most of the current works [32, 33, 15]. However, in reality, there are also some *Non-Boolean Variability Models*, which contain constraints in the form of data domains including integer or float numbers, or strings, in addition to Boolean expressions. For instance, a feature may require an arbitrary assignment determined from a set of enumerations or a given range. Passos et al. did an initial analysis of the 116 Non-Boolean Variability Models in eCos [34]. They characterized the types of Non-Boolean features, the kinds and quantities of the Non-Boolean constraints in use. This work provided a very good start for the research on the configuration of Non-Boolean or the mix of Boolean and Non-Boolean Variability Models.

The third limitation is introduced by the nature of multi-objective optimization algorithms. The optimization is a process of searching the approximate solutions to find what can be considered very close to the "correct" valid ones. However, it is impossible to promise the existence of the valid solutions for all feature models in feature configuration. Some feature models are void, which represent no products [9]. A series of products may be formed in the valid feature models, but the extra functional and nonfunctional requirements probably result in no products being created. To avoid "blind" searching, model validation before the optimization selection is necessary. Our work and several other works [35, 36, 37] are concentrated only on the optimization process after model checking.

8. Related Works

In this section we discuss the related research on optimized feature selection. The related methods can be classified into two groups: exact and approximate optimization for SPL configuration.

Exact optimization approaches are always based on solving a *Constraint Satisfaction Problem (CSP)* to find the final configurations. Benavides et al. transformed the extended feature model with extra-functional features into a CSP, and then used constraint programming to do the related reasoning [33]. In addition to basic reasoning, a CSP solver can also find the solutions that minimizes or maximizes the objective function. For instance, Benavides et al. implemented this using OPL Studio, a commercial CSP Solver [33]. Siegmund et al. provided a holistic approach called SPL Conquer which supports the definition of non-functional properties, the measurement of properties and the search for optimal configurations using a CSP solver [32]. This work provides an interface for stakeholders to aggregate the multiple objectives into one, according to their preferences on various properties. White et al. did the research on multi-step configuration process and developed an automated method MUSCLE to derive optimal selections based on CSP solvers [38]. The NFRs between two successive configurations were a consideration. These approaches focus on single-objective optimization. The multiple objectives are integrated into a single one by the corresponding weights of each objective, which are always assigned by customers [32]. However, it isn't always easy to find the exact numbers to present the relative importance of the quite different objectives. Olaechea et al. tried an exact way Guided Improvement Algorithm (GIA) for multi-objective optimization on five case models [39]. They showed that GIA works well on smaller feature models with up to 44 features. But it fails for large models with hundreds of features like *EShop* with 290 features. In our case, the smallest model *eCos* has 1244 features. Thus, although exact optimization approaches can generate the solutions with high accuracy, the computation cost is too expensive. These existing exact approaches don't suit for our problem.

Due to the exponential computing time of exact approaches, many researchers explored various approximate approaches. Guo et al. presented an approach named GAFES based on a genetic algorithm towards the optimization on resource use [40]. It controlled the violation of resource constraints through a penalty function, depicting on the ratio of objective function value to the consumed resources. By using a group of weights, multiple resource consumption is accumulated into one objective. White et al. designed a polynomial time approximation algorithm, called Filtered Cartesian Flattening (FCF), which transforms an optimal feature selection problem with resource constraints into a multi-dimensional multi-choice knapsack problem (MMKP), which then gets solved with an MMKP approximation algorithm [10]. Sion et al. presented a reference architecture of product configuration tooling, assisting engineers to gain more insights about the process of product configuration, the interactions between features and quality concerns [41]. Soltani el al. tried a method for solving a planning problem to find optimal feature configurations [12, 11]. One important consideration is the user's preference on the non-functional properties. Firstly, they transformed a feature model and stakeholders' preference on various non-functional properties into a planning problem. Then a framework was proposed to select features using Hierarchical Task Network (HTN) planning techniques. The identification of an optimal plan was performed by SHOP2, an HTN-based planning system. In this work, the multiple objectives were integrated based on the relative preferences on properties and the interdependency between non-functional properties. However, they tested their proposed solution with smaller feature models containing 200 features at most. Stein et al. also discussed the feature selection in the face of multiple stakeholders' preference [42]. Different from Soltani's scenarios in which the discrepancy in multiple stakeholders' preferences is not discussed and the preferences of qualities can be expressed with relative numbers, Stein et al. classified the preferences of stakeholders into hard and soft constraints, and used seven different strategies for choosing the optimal configuration considering a group of stakeholders [42]. These approaches also focus on single-objective optimization. Just as the discussion above, for the competing and conflicting multiple objectives, their relative importance isn't always available. It is therefore impossible to integrate these objectives into a single one. So we have to investigate the approximate approaches for multi-objective optimization.

To the best of the authors' knowledge, Sayyad et al. firstly explored multi-objective evolutionary algorithms to find the optimal solutions towards multiple competing and conflict objectives [2]. After experimentation, they showed that IBEA outperforms other common MOEAs. Specifically, with the increasing number of objectives, IBEA presents more noticeable advantages. To scale the algorithm for larger feature models, Sayyad et al. introduced some brilliant heuristics into IBEA [3]. Through experiments, it was shown that the improved IBEA works on the largest realistic feature model, *Linux*, containing 6,888 features. However, feature-conformance was not explicitly dealt with, which is essential for feature selection. On the basis of Sayyad's work, Henard et al. proposed a further improved algorithm called SATIBEA [15]. In this work, Henard et al. combined IBEA and a CSP solver. The core design is two smart operators: mutation and replacement. They showed SATIBEA gained significant outperformance over the improved IBEA on five large-scale feature models. SATIBEA and the improved IBEA are therefore regarded as the current state-of-the-art in multi-objective optimization approaches for SPL configuration.

Previously, we designed a Multi-Objective Optimization algorithm template for Feature selection (MOOFs) to automatically select features under functional and non-functional requirements [43]. A reviser was proposed to merge into MOOFs to fix every invalid solution. MOOFs can promise all optimal solutions conform to the feature model. Unfortunately, with further research, we found that the frequent correction reduces the search-space too much and leads to the low diversity of the final approximate Pareto Fronts. Therefore, extending search space during optimization is critical for the process improvement. Based on this, we proposed IVEA [1]. Rather than conducting the fixes directly, we filtered the invalid solutions gradually, which violates more constraints than a *progressively smaller threshold*. A bigger threshold at the beginning is beneficial for preserving the solutions which violate some constraints but have better optimization objectives. Moreover, these solutions have potential to create sound offspring. After filtering, the solutions with better multiple objectives are kept. This is the idea of our 2D fitness function. Experiments showed that this novel fitness function can improve the search process significantly when testing on two feature models. However, when running IVEA on *FM2000*, which is a large feature model in SPLOT, some invalid solutions violating a certain number of constraints (usually much less than the average violations of the initial population) would be created circularly which creates an infinite loop. This made it clear that the fitness function is not enough for the selection process on large feature models. That's the reason that we want to improve IVEA to enhance its scalability.

9. Conclusion and Future Work

The goal of this paper was to improve our existing methods [43, 1] of optimized feature selection for configuring products in large feature models. To this end we proposed a novel multi-objective optimization algorithm, IVEA-II, through reserving the 2D fitness function in our previous work [1], introducing crowding distance [5] and designing a new mutation operator. For the scalability evaluation purpose, the experiments of this paper were performed on five objectives optimization with five large feature models.

By comparing our IVEA-II with the other two leading algorithms, we have shown IVEA-II outperforms the others on the efficiency of generating the optimal valid solutions and the qualities of these solutions. We have also shown the positive impacts of each feature of IVEA-II on optimized feature selection through experiments.

As far as we know, there is no publicly available data about the properties of features in realistic projects which can be used as references for feature selection. We used previous data which was created manually [3, 15]. In future work, we will collect the relevant data based on the traceability between features and the related artifacts. For instance, through tracing features to their component implementations, we can record resource consumption, such as the footprint of each feature. It is also possible to get the data related to reliability, such as the error times that occur in the components of each feature. We believe this data will be helpful for all research on optimized feature selection.

10. Acknowledgement

The work in this present paper is funded by the National Science Foundation of China No.61672078 and State Key Laboratory of Software Development Environment No.SKLSDE-2016ZX-10. The studying of the first author Xiaoli Lian in DePaul University is funded by China National Scholarship Council.

References

[1] X. Lian, L. Zhang, Optimized feature selection towards functional and non-functional requirements in software product lines, in: Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, 2015, pp. 191–200.

[2] A. S. Sayyad, T. Menzies, H. Ammar, On the value of user preferences in search-based software engineering: A case study in software product

lines, in: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 492–501.

[3] A. Sayyad, J. Ingram, T. Menzies, H. Ammar, Scalable product line configuration: A straw to break the camel's back, in: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, 2013, pp. 465–474.

[4] E. Zitzler, S. Kunzli, Indicator-based selection in multiobjective search, in: in Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII), Springer, 2004, pp. 832–842.

[5] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, Evolutionary Computation, IEEE Transactions on 6 (2) (2002) 182–197.

[6] E. Zitzler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm, Tech. rep. (2001).

[7] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-oriented domain analysis (foda) feasibility study, Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1990).

[8] M. Mendonça, T. T. Bartolomei, D. Cowan, Decision-making coordination in collaborative product configuration, in: Proceedings of the 2008 ACM Symposium on Applied Computing, SAC '08, ACM, New York, NY, USA, 2008, pp. 108–113.

[9] D. Benavides, S. Segura, A. Ruiz-Cortes, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (6) (2010) 615 – 636.

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

[11] M. Asadi, S. Soltani, D. Gasevic, M. Hatala, E. Bagheri, Toward automated feature model configuration with optimizing non-functional requirements, Information and Software Technology 56 (9) (2014) 1144–1165.

[12] S. Soltani, M. Asadi, D. Gasevic, M. Hatala, E. Bagheri, Automated planning for feature model configuration based on functional and non-functional requirements, in: 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2-7, 2012, Volume 1, 2012, pp. 56–65.

[13] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, S. Apel, G. Saake, Spl conqueror: Toward optimization of non-functional properties in software product lines, Software Quality Journal 20 (3-4) (2012) 487–517.

[14] H. Eskandari, C. Geiger, G. Lamont, Fastpga: A dynamic population sizing approach for solving expensive multiobjective optimization problems, in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Evolutionary Multi-Criterion Optimization, Vol. 4403 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 141–155.

[15] C. Henard, M. Papadakis, M. Harman, Y. Le Traon, Combining multi-objective search and constraint solving for configuring large software product lines, in: Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, Vol. 1, IEEE, 2015, pp. 517–528.

[16] J. J. Durillo, A. J. Nebro, jmetal: A java framework for multi-objective optimization, Advances in Engineering Software 42 (10) (2011) 760 – 771.

[17] E. H.-C. Jackson, Daniel, D. Rayside, The guided improvement algorithm for exact, general-purpose, many-objective combinatorial optimization, 2009.

[18] S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki, Reverse engineering feature models, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, ACM, New York, NY, USA, 2011, pp. 461–470.

[19] T. Berger, S. She, R. Lotufo, K. Czarnecki, Variability modeling in the systems software domain, Tech. rep. (2012).

[20] A. Arcuri, L. C. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011, 2011, pp. 1–10.

[21] A. Sayyad, H. Ammar, Pareto-optimal search-based software engineering (posbse): A literature survey, in: Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on, 2013, pp. 21–27.

[22] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, V. da Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, Evolutionary Computation, IEEE Transactions on 7 (2) (2003) 117–132.

[23] D. A. V. Veldhuizen, G. B. Lamont, Multiobjective evolutionary algorithm research: A history and analysis (1998).

[24] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE transactions on Evolutionary Computation 3 (4) (1999) 257–271.

[25] T. Hawkins, Lebesgue's theory of integration: its origins and development 282.

[26] O. J. Dunn, Multiple comparisons among means, Journal of the American Statistical Association 56 (293) (1961) 52–64.

[27] A. Vargha, H. D. Delaney, A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong, Journal of Educational and Behavioral Statistics 25 (2) (2000) pp. 101–132.

[28] L. Linsbauer, E. R. Lopez-Herrejon, A. Egyed, Recovering traceability between features and code in product variants, in: Proceedings of the 17th International Software Product Line Conference, ACM, 2013, pp. 131–140.

[29] R. Tsuchiya, T. Kato, H. Washizaki, M. Kawakami, Y. Fukazawa, K. Yoshimura, Recovering traceability links between requirements and

source code in the same series of software products, in: Proceedings of the 17th International Software Product Line Conference, ACM, 2013, pp. 121–130.

[30] W. Ji, T. Berger, M. Antkiewicz, K. Czarnecki, Maintaining feature traceability with embedded annotations, in: Proceedings of the 19th International Conference on Software Product Line, ACM, 2015, pp. 61–70.

[31] P. Valov, J. Guo, K. Czarnecki, Empirical comparison of regression methods for variability-aware performance prediction, in: Proceedings of the 19th International Conference on Software Product Line, ACM, 2015, pp. 186–190.

[32] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, S. Apel, G. Saake, Spl conqueror: Toward optimization of non-functional properties in software product lines, Software Quality Journal 20 (3-4) (2012) 487–517.

[33] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: Seminal Contributions to Information Systems Engineering, Springer, 2013, pp. 361–373.

[34] L. Passos, M. Novakovic, Y. Xiong, T. Berger, K. Czarnecki, A. Wąsowski, A study of non-boolean constraints in variability models of an embedded operating system, in: Proceedings of the 15th International Software Product Line Conference, Volume 2, SPLC '11, ACM, New York, NY, USA, 2011, pp. 2:1–2:8.

[35] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, M. Toro, Automated error analysis for the agilization of feature modeling, Journal of Systems and Software 81 (6) (2008) 883–896.

[36] D. Batory, Feature models, grammars, and propositional formulas, Springer, 2005.

[37] M. Mendonca, A. Wąsowski, K. Czarnecki, Sat-based analysis of feature models is easy, in: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, 2009, pp. 231–240.

[38] J. White, B. Dougherty, D. C. Schmidt, D. Benavides, Automated reasoning for multi-step feature model configuration problems, in: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, 2009, pp. 11–20.

[39] R. Olaechea, D. Rayside, J. Guo, K. Czarnecki, Comparison of exact and approximate multi-objective optimization for software product lines, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 92–101.

[40] J. Guo, J. White, G. Wang, J. Li, Y. Wang, A genetic algorithm for optimized feature selection with resource constraints in software product lines, Journal of Systems and Software 84 (12) (2011) 2208 – 2221.

[41] L. Sion, D. Van Landuyt, W. Joosen, G. de Jong, Systematic quality trade-off support in the software product-line configuration process, in: Proceedings of the 20th International Systems and Software Product Line Conference, ACM, 2016, pp. 164–173.

[42] J. Stein, I. Nunes, E. Cirilo, Preference-based feature model configuration with multiple stakeholders, in: Proceedings of the 18th International Software Product Line Conference-Volume 1, ACM, 2014, pp. 132–141.

[43] X. Lian, L. Zhang, An evolutionary methodology for optimized feature selection in software product lines, in: The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013., 2014, pp. 63–66.

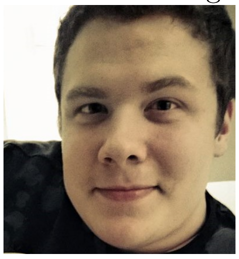
Biography

Xiaoli Lian is a PhD student in Beihang University at Beijing, China. Her research interests are on the software qualities.

Previously, she did some research on quality validation in the design phrase of embedded system. She mainly focused on time-related attributes. She is also interested in the quality satisfaction in software product line engineering. She did exploration on the feature selection towards multi-objective optimization during software derivation in software product lines. At the same time, she explores the maintenance of quality-related architectural design during source code change by detecting and tracing architectural designs cooperating with Pro. Jane Cleland-Huang in DePaul University.

Li Zhang is a full professor in School of Computer Science and Engineering, at Beihang University in Beijing, China. She received her Bachelor, Master and PhD degrees in School of Computer Science and Engineering, Beihang University, in 1989, 1992 and1996, respectively. Her main research area is software engineering, with specific interest in requirements engineering, software/system architecture, model-based engineering, model-based product line engineering and empirical software engineering. She is a committee member of Software Engineering in CCF (China Computer Federation) and vice chair of Education Committee in CCF. She was the PC chair of ICSSP 2013 and have been PC member for several international academic conference, such as ICSSP, IESA, APRES, and etc.

Jing Jiang is an assistant professor in the State Key Laboratory of Software Development Environment of Beihang University, Beijing, China. Her research interests include software engineering, empirical software engineering, data mining, human factors and social aspects of software engineering. She received her B.S. and Ph.D. degrees in computer science from Peking University, Beijing, in 2007 and 2012, respectively.



William Goss is an aspiring researcher working at DePaul University. Currently, he works under Professor Jane Cleland-Huang on research that is interested in software traceability. This work consists of tracing software and their requirements across different versions of code, and another project that looks at translating natural language queries into informative SQL queries that pull useful information from software documents and code. Outside of software engineering, he is interested in machine learning and how to apply generalizations across complex data.