An Evolutionary Methodology for Optimized Feature Selection in Software Product Lines

Xiaoli Lian  LiZhang
State Key Laboratory of Software Development Environment
School of Computer Science and Engineering
Beihang University
Beijing, China
lianxiaoli@cse.buaa.edu.cn  lily@buaa.edu.cn

Abstract—Feature modeling is the primary technology to capture and document the commonalities and variability among all of the members in a product line. Individual products are customized by selecting features according to the requirements. The work of feature selection is complex because of: 1) the complex dependencies and constraint relationship amongst features; 2) the multiple competing and conflicting non-functional requirements (NFRs); 3) the constraints to NFRs; 4) the explicit functional requirements. To select optimized feature set that conforms to the feature relations and satisfies both the functional and non-functional requirements and the related constraints, an evolutionary algorithm template which employs multi-objective optimization algorithms to optimally select features in SPLs, is proposed. In the experiments, two different algorithms are designed based on our template. Empirical results show the remarkable performance of our algorithms on time especially when the feature models are large and complex.

Keywords—Product Line Engineering; feature selection; multi-objective optimization; non-functional optimization

I. INTRODUCTION

Product Line Engineering[1] has been shown an effective approach to product software for decreasing the cost, improving the product quality and accelerating time to market by many organizations such as Boeing, Bosch Group, Nokia and so on. Feature model is a popular way to express the commonalities and variability among all of the members in a product family. The tailored products are derived by selecting and removing features from feature model.

In realistic feature model, non-functional requirements (NFRs) are always demanded besides the functional ones (FRs). These NFRs are also conflicting and have to be traded-off. In addition, massive features have constraints and dependencies relation with other ones, e.g., 86% features declare constraints in Linux and eCos. So a newly selected feature would often result some other ones obsolete and these have to be reselected.

Multiobjective Evolutionary Optimization Algorithms (MEOAs) aims to optimize more than one objective in the presence of trade-offs. It is a proper way to deal with the competing NFRs in feature selection. Unfortunately, the random crossover and mutation of MEOAs inevitably destroy the feature dependencies and constraints. So in the current paper, a repair operator is designed to revise the destructive effects by the random operations during evolutionary process.

The main contributions of this paper can be summarized as: (1) We design a uniform representation for different kinds of relations amongst features; (2) We design SolutionRevise, a common solution revise operator, which can be applied to different MEOAs and revise the feature selection that violates the feature model constraints; (3) We design an evolutionary algorithm template named MOOFs and show the remarkable performance by two algorithms named MOOF_m and MOOF_n.

II. FEATURE MODEL

FIG.1 illustrates a simple feature model for mobile phone product line. The rectangles indicate features and the line between them are their relations. From FIG.1, we can see two basic kinds of relations amongst features:

1. Relationships between a parent feature and its child features, includes mandatory, optional, alternative and or.
2. Cross-tree (or cross-hierarchy) constraints typically state that one feature excludes or requires another one.

![FIG. 1 FEATURE MODEL FOR MOBILE PHONE PRODUCT LINE](image)

There can be group cardinality of the form [m,n] in feature groups, which declare that at least m and at most n members of a feature group can be selected. Cross-tree constraint ratio (CTCR) is defined in [2] as the ratio of the number of features in the cross-tree constraints to the number of features in the feature tree and it is a common indicator to indicate the complexity of a feature model.

To deal with the NFRs, some researches extended the feature model with feature attributes[2][3][4][5]. In our paper, we assumed...
the NFRs have been quantified. The classification and quantification of the NFRs are not our concerns in this paper.

III. METHODOLOGY

A. Preprocessing

1) Feature model constraints and functional requirements representation and non-functional requirements

From section II, we can see there are different kinds of relations amongst features. These different relations restrict the existence of features in different way. To simplify the treatment for different constraints, we encode them in a uniform format firstly. We defined them as rule and the Chomsky grammar can be seen in Definition 1.

Definition 1. A rule is defined as a quadri-tuple (N, Σ, P, S), where the non-terminal set N = { rule, A, B, G, Min, Max, N, N*}, the terminal set Σ = {<, >, [ , ], *, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, featureID }, the start symbol S = { rule}. The following are the rewrite rules in P.

\[
\begin{align*}
\text{rule} & \Rightarrow A, B > \\
A & \Rightarrow e | B \\
B & \Rightarrow \text{featureID} | G \\
G & \Rightarrow [ \text{featureID}, \text{featureID} ] | [ \text{Min}, \text{Max} ] \\
\text{Min} & \Rightarrow 0 | N, N* \\
\text{Max} & \Rightarrow \text{Min} | * \\
N & \Rightarrow 0, 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \\
N_+ & \Rightarrow 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
\end{align*}
\]

A rule can be expressed as <A, B> which is a relation that A requires B. A and B are marked first and second respectively in the following sections. Both A and B can be a single feature or a feature group. Taking one ternary cross-tree constraint \( \sim a \) or \( \sim b \) or \( c \) for example, the rule can be expressed as \( \langle a, b \rangle [2,2], c \rangle \) which means that \( c \) must be contained when both \( a \) and \( b \) are selected.

In consideration of clear expression, we state the condition that a feature group is satisfied.

Definition 2. A feature group is satisfied when the number of feature members selected is between \( \minRange \) and \( \maxRange \).

In addition, A can be \( e \) when B must be satisfied in any case. Through analyzing the explicit functional requirements, we found they can be described as rule with the \( e \) first. If one feature noted as \( f \) is required, there should be a rule \( < e, f > \) which means that \( f \) must be selected anyhow.

The feature selection conforming to all of the rules is called as “correct” solution in this paper. Otherwise, the selection is “incorrect”.

Just as the discussion in section II, we assumed the NFRs have been quantified by the attributes of features and the values have been assigned in some way. The constraints for NFRs were described as equalities or inequalities, e.g., cost ≤ 400.

2) Feature chromosome encoding

The feature chromosome is encoded as binary string. The length of the binary string is equal with feature number. If one feature is selected, the corresponding bit is 1; otherwise the bit set is 0.

B. MOOFs: an Multi-Objective Optimization algorithm template for Feature selection

Based on the idea of MEOAs, we designed our algorithm template MOOFs listed in Algorithm 1. In MOOFs, we did not confine any concrete method such as population initialization, the fitness function, the crossover operator, and the mutation operator and so on. The selection of these operators depends on the certain application.

Algorithm 1: MOOFs

\[
\begin{align*}
\text{Input: } & \ a \ (\text{population size}) \\
& \ N \ (\text{maximum number of generations}) \\
& \ k \ (\text{fitness scaling factor}) \\
& \ R \ (\text{rule set}) \\
\text{Output: } & \ A \ (\text{Pareto set approximation}) \\
\text{Initialize } & \ P \\
& \ p^* = \text{SolutionRevise}(p), \text{for all } p \in P \\
\text{Repeat} \\
& \text{calculate fitness values of } P^* \\
& \text{select two individuals from } P^* \text{ to fill mating pool } M \\
& \text{apply mutation operator to } M \text{ and get offspring } O \\
& \text{O}^* = \text{SolutionRevise}(O) \\
& P^* \Leftarrow O^* \\
& \text{Select and remove individuals from } P^* \\
& \text{Stop until } m \geq N \text{ or other stopping condition} \\
& \text{Return } A \text{ that is the Pareto Front approximation in } P^*
\end{align*}
\]

C. SolutionRevise: an algorithm to revise feature selections

According to the type of first and second in a rule defined in section III.A.1), the rules can be grouped into five categories: \( < v, \text{ single feature} > \), \( < v, \text{ feature group} > \), \( < \text{single feature}, \text{ single feature} > \), \( < \text{single feature}, \text{ feature group} > \) and \( < \text{feature group}, \text{ single feature} > \). The type \( < \text{feature group}, \text{ feature group} > \) does not exist, because a feature group must have a single feature parent. So their relationships can be depicted as \( < \text{single feature}, \text{ feature group} > \). On the basic of this classification, we defined two principles to select or remove a feature.

Algorithm 2: SolutionRevise

\[
\begin{align*}
\text{Input: } & \ S \ (\text{one configuration to be revised}) \\
& \ R \ (\text{rule set}) \\
\text{Output: } & \ S_f \ (\text{feature selection conforming to } R) \\
& \ S_f \Leftarrow \varnothing, S_f \Leftarrow \varnothing, S_f \Leftarrow \varnothing \\
\text{foreach } & \text{rule } r \text{ in } S \\
& \text{if the second marked as } s \text{ of } r \text{ is a group and the } \minRange > 0, \text{ then} \\
& \text{S}_f \Leftarrow r \\
\text{end} \\
& \text{IncludeFeature(0);} \\
& \text{foreach } \text{feature } f \text{ in } S \\
& \text{if } f \neq S_f \text{ and } f \notin S_f, \text{ then IncludeFeature(f)} \\
\text{end} \\
& \text{foreach } \text{rule } r \text{ in } S_f \\
& \text{Note the number of feature members in the second of } r \text{ which has been selected as } N \\
& \text{While } N < \minRange \text{ do} \\
& \text{select feature } f'' \neq S_f \text{and } f'' \notin S_f \\
& \text{IncludeFeature(}f''\text{)} \\
& \text{Recount } N \\
\text{end} \\
\text{Return } A \text{ that is the Pareto Front approximation in } P^*
\end{align*}
\]

Definition 3: A feature \( f \) is selected, if any of the following conditions is true: 1) \( f \) is neither included nor excluded; 2) if \( f \) as a single feature is the second of a rule and the corresponding first that is also a single feature is selected; 3) if \( f \) as a single feature is the second of a rule and the corresponding first is a feature group
and the first is satisfied. 4) if existing in a feature group which is the second of a rule and the first is null or a single feature being selected, the number of group members that are not excluded is just the minRange of the feature group.

**Algorithm 3: IncludeFeature**

**Input:** f (one feature to be included)  
R (rule set)  
S (the set of rule whose second dimension is a feature group with the minRange >0)

**Output:** S, S

foreach rule r in S

Denote the current group of r as S

if f ∈ S and the first of r is satisfied or the first is null then

Count the feature number of S in S as Y

if maxRange = Y then

ExcludeFeature (the other features of S both in nor S)

end

foreach rule r in R

if the first is a group set S and f ∈ S and the first is satisfied and the second is a single feature, then IncludeFeature(f)

if the first is f and the second is a single feature then

IncludeFeature(f)

if the first is f and the second is a group set S then

set the feature number of S in S as Y, the feature number in S as E

if maxRange = Y then

ExcludeFeature (the other features of S both in nor S)

if size(S) - E = minRange then

IncludeFeature (other features of S neither in nor S)

end

**Algorithm 4: ExcludeFeature**

**Input:** f (one feature to be excluded)  
R (rule set)  
S (the set of rule whose second dimension is a feature group with the minRange >0)

**Output:** S, S

foreach rule r in S

Denote the current group of r as S

if f ∈ S and the first of r is satisfied or the first is null then

Count the feature number of S in S as E

if size(S) - E = minRange then

IncludeFeature (other features of S neither in nor S)

end

foreach rule r in R

if as a single feature is the second of r, then ExcludeFeature(f)

end

**Definition 4:** A feature f is excluded, if any of the following conditions is true: 1) if as a single feature is the first of a rule, the second which is also a single feature is excluded. 2) if as a single feature is the first of a rule, and the second is a group feature and the number of feature members which has not been excluded is less than minRange. 3) if existing in a feature group which is the first of a rule, the second which is a single feature is excluded, and the number of features that are not excluded is above the lower limit of the group. 4) if existing in a feature group which is the second of a rule and the first is null or a single feature being selected, the number of features included is just the maxRange of the group.

**SolutionRevise** was designed based on these two principles. Algorithm 2 shows the detailed process of SolutionRevise. SolutionRevise contains two related operators named IncludeFeature and ExcludeFeature. These two algorithms, shown in Algorithm 3 and Algorithm 4, determine whether a feature is selected or excluded. IncludeFeature puts one specific feature f and the features that require f in SI and excludes the extra features when any maxRange is achieved after including f. Similarly, the operator of ExcludeFeature puts one specific feature f and the features that require f in SE and puts the left features in SI when the number of features in any group left by SE is minRange after excluding f.

**IV. EXPERIMENT SET UP**

**A. Setting up the feature model**

Four feature models including two realistic models and two synthesis were selected from SPL*2 (7), a popular feature model repository. In this paper, we augmented the feature models with the same attributes and contributions of their values as that in [6]. And the five optimization objectives[6]: to minimize total cost, to maximize features that were used before, to minimize the total number of known defects, to maximize correctness and to maximize the number of offered features.

**TABLE I. FEATURE MODELS USED IN THIS PAPER**

<table>
<thead>
<tr>
<th>Feature Model</th>
<th>Features</th>
<th>Total Rules</th>
<th>CTCs</th>
<th>CTCR</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebProtal</td>
<td>43</td>
<td>51</td>
<td>6</td>
<td>2.6%</td>
<td>COST&lt;400</td>
</tr>
<tr>
<td>EShop</td>
<td>290</td>
<td>333</td>
<td>21</td>
<td>6.8%</td>
<td>COST&lt;2900</td>
</tr>
<tr>
<td>FM500</td>
<td>500</td>
<td>572</td>
<td>50</td>
<td>7.8%</td>
<td>COST&lt;4000</td>
</tr>
<tr>
<td>FM1000</td>
<td>1000</td>
<td>1146</td>
<td>100</td>
<td>8.0%</td>
<td>COST&lt;9000</td>
</tr>
</tbody>
</table>

**B. Setting up algorithms**

In this paper, we designed MOOF_r and MOOF_s, by introducing our SolutionRevise operator into IBEA suggested by [6]. Our two algorithms with NSGAII[8], SPEA2[9], IBEA_r[10] and IBEA_s[10] were experimented on the four models. All of these algorithms were implemented in jMetal[13], a popular framework for multi-objective optimization. The parameters of these algorithms are default values in jMetal: population size=100, single-point crossover, crossover probability=0.90, bit-flip mutation, mutation probability=1/NumberOFVariables.

To diminish the impact of stochastic behavior of these randomized algorithms, we ran every algorithm independently run 30 times on a feature model. Initially, we performed all algorithms with 25K objective function evaluations. If no “correct” solutions were created, the evaluations time would turn to 500K. If not still, the evaluations would be 1M or 2M or 5M or 10M. But if no still even with 10M evaluations, the trial would be stop.

**V. EMPIRICAL RESULTS**

This section shows the comparing results of six algorithms on four feature models. We compare the run time to generate one correct solution.
All other algorithms perform moderately with smaller WebProtal and EShop. However, to bigger model FM500, only IBEA_{HD} can get correct solutions when evaluations is 500K. To FM1000, no correct solution is generated by all these algorithms till 10M objectives functional evaluations. However, the SolutionRevise operation makes MOOF_{HD} and MOOF_{r^+} always produce correct solution.

<table>
<thead>
<tr>
<th>Feature Model</th>
<th>Algorithms</th>
<th>Eval.</th>
<th>RunTime</th>
<th>STR</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebProtal</td>
<td>NSGAII</td>
<td>25K</td>
<td>0.55sec</td>
<td>3.23</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>25K</td>
<td>59.08sec</td>
<td>453.46</td>
</tr>
<tr>
<td></td>
<td>IBEA_{HD}</td>
<td>25K</td>
<td>0.11sec</td>
<td>-0.15</td>
</tr>
<tr>
<td></td>
<td>IBEA_{r^+}</td>
<td>25K</td>
<td>21.73sec</td>
<td>166.15</td>
</tr>
<tr>
<td></td>
<td>MOOF_{HD}</td>
<td>25K</td>
<td>0.80sec</td>
<td>5.15</td>
</tr>
<tr>
<td></td>
<td>MOOF_{r^+}</td>
<td>25K</td>
<td>0.13sec</td>
<td>/</td>
</tr>
<tr>
<td>EShop</td>
<td>NSGAII</td>
<td>500K</td>
<td>47.22sec</td>
<td>261.33</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>500K</td>
<td>3.67sec</td>
<td>19.39</td>
</tr>
<tr>
<td></td>
<td>IBEA_{HD}</td>
<td>25K</td>
<td>3.68sec</td>
<td>19.44</td>
</tr>
<tr>
<td></td>
<td>IBEA_{r^+}</td>
<td>50K</td>
<td>135.97sec</td>
<td>754.39</td>
</tr>
<tr>
<td></td>
<td>MOOF_{HD}</td>
<td>25K</td>
<td>1.88sec</td>
<td>9.44</td>
</tr>
<tr>
<td></td>
<td>MOOF_{r^+}</td>
<td>25K</td>
<td>0.18sec</td>
<td>/</td>
</tr>
<tr>
<td>FM500</td>
<td>NSGAII</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>IBEA_{HD}</td>
<td>500K</td>
<td>11.45sec</td>
<td>44.80</td>
</tr>
<tr>
<td></td>
<td>IBEA_{r^+}</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>MOOF_{HD}</td>
<td>25K</td>
<td>2.67sec</td>
<td>9.68</td>
</tr>
<tr>
<td></td>
<td>MOOF_{r^+}</td>
<td>25K</td>
<td>0.25sec</td>
<td>/</td>
</tr>
<tr>
<td>FM1000</td>
<td>NSGAII</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>IBEA_{HD}</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>IBEA_{r^+}</td>
<td>10M</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>MOOF_{HD}</td>
<td>25K</td>
<td>4.74sec</td>
<td>9.68</td>
</tr>
<tr>
<td></td>
<td>MOOF_{r^+}</td>
<td>25K</td>
<td>0.27sec</td>
<td>/</td>
</tr>
</tbody>
</table>

\*\*\* means no correct solution produced by the algorithm in the corresponding row and \*/\* means no calculation on STR. If the STR is negative, the time consumed by MOOF_{r^+} is larger than the algorithm in the row.

From the column of RunTime, we can observe that IBEA_{HD} works best comparing with NSGAII, SPEA2 and IBEA_{r^+}, which is consistence with [6]. However, to larger model, its performance is turning worse than our algorithms.

Looking at the column of STR, except for the smallest WebProtal model, about 3 to 654 times more seconds are consumed by other algorithms to produce one correct solution than MOOF_{r^+}.

VI. RELATED WORK

Some remarking works transformed the feature selection into a SAT problem like Benavides et al.[2]. White et al.[12] and Guo et al. [13] selected features satisfying a series of resource constraints. Sayyad et al.[6] tried MEOAs to do feature selection towards multi-attribute optimization and proved the significant of IBEA. However, these approaches did not concern the explicit FRs and NFRs with constraints or not.

A solution repair operator was also proposed in [13]. It is for the basic feature model with binary cross-tree constraints. Mendonca et al.[14] observed that the cross-tree constraints in the realistic models consisted of a mix of binary and ternary CNF clauses which are considered in our methodology.

VII. CONCLUSIONS AND FUTURE WORKS

Aiming to automatically select features that conforms to the feature model and satisfies both the functional and nonfunctional requirements and the related constraints, we provided a multi-objective optimization algorithm template for feature selection which combined our configuration reviser with the existing MEOAs flexibly. During empirical experiments, we designed two algorithms based on the IBEA suggested by [6]. Through comparing with four widespread MEOAs, our algorithms work best when the timing-consuming and multi-attribute performance are considered comprehensively.

REFERENCES