Available online at www.sciencedirect.com

ScienceDirect

NasBASE 2015 00 (2015) 1–16

NasBASE 2015

First North American Search Based Software Engineering Symposium

SPLRevO: Optimizing Complex Feature Models in Search Based Reverse Engineering of Software Product Lines

Thammasak Thianniwet and Myra B. Cohen

Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE, 68588-0115 USA

Abstract

Recent work to reverse engineer feature models from a set of products leverages a genetic algorithm. This idea has been realized as a framework called ETHOM, however, it suffers from two limitations. First, the existing fitness functions either do not penalize models that contain products which are not in the original dataset, or create models which are missing too many valid products. This results in models that greatly over or under-approximate the feature space. Second, the ETHOM framework only supports models with simple cross-tree constraints in the form of binary excludes or requires, meaning that feature models with complex cross-tree constraints may not be captured precisely. In this work, we present an improved fitness function that more effectively guides the search to balance both additional and missing products. In addition, we have created a new framework, called SPLRevO, containing a chromosome and additional mutation and crossover operators to support complex cross-tree constraints. In a study spanning 99 feature models of various sizes, both with and without complex cross-tree constraints, we show that our new fitness within the ETHOM framework improves the F-measure over the existing fitness functions from 7 to 50%. When the new fitness is used within SPLRevO, we increase the F-measure over ETHOM by 20 to 63%.

Keywords: genetic algorithms, software product lines, reverse engineering

1. Introduction

Software product lines (SPLs) are families of related software products, composed of sets of common and variable features, defined and managed by a feature model [1]. Feature models represent the set of all possible products within an SPL, specifying how the common and variable features of the product can be combined [2–4]. A feature model consists of two parts: (1) a feature diagram, the graphical notation for the core set of products in the feature model, and (2) a (possibly empty) set of additional cross-tree constraints, which are constraints that describe dependencies between elements of the feature model, but which cannot be easily expressed graphically [5]. Both the feature diagram and the cross-tree constraints can be represented as a set of propositional formulas, therefore it is possible to determine whether or not a particular product composed of a set of features is a valid member of the product family through satisfiability analysis [6]. Feature models are used to guide product creation, product maintenance and can also be used to enhance and guide product line testing [7–10].

Email address: {tthianni,myra}@cse.unl.edu (Thammasak Thianniwet and Myra B. Cohen)

Modifying an existing software product line, or designing a system test plan, without consideration of the feature model may result in wasted effort due to the inclusion of invalid products [5]. However, feature models often do not exist [11]. Manually creating feature models from existing software is time-consuming and requires significant effort from a modeler. Therefore, recent work has moved towards automated feature model extraction from existing artifacts, such as source code, documents, feature names, and lists of the valid products [5, 11–17]. For instance, recent work by She et al. proposes procedures for reverse engineering feature models from existing feature dependencies and descriptions [5]. However, feature dependencies and their descriptions may not exist for some systems. Greevy also proposes reverse engineering feature models by analyzing run-time behavior, structure, and the dependencies of features [12]. This approach requires exhaustive analysis of the source code (which may be inaccessible) in combination with dynamic analysis which can be time and memory intensive.

Recently, search-based approaches have been applied to this problem [13, 18, 19], ,and to the domain of software product lines in general [20]. Lopez-Herrejon et al. proposed an evolutionary algorithm, *Evolutionary algoriTHm for Optimized feature Models* (or ETHOM), for reverse engineering feature models from a set of known valid products (an individual product is also called a feature set in that work) [13, 18]. The empirical evaluation shows promise and an associated tool has been released [13]. Related to this idea is the use of genetic programming to synthesize feature models [19]. An orthogonal approach, using a graph based heuristic algorithm to directly construct models from the set of propositional formulas has also been proposed [5, 21]. This differs from the search based approach which uses the formulas only to enumerate the valid set of products, but uses the actual set of products as the oracle for feature model correctness. In practice, the set of equations may be harder to obtain than a set of valid products. Work by Nadi et al. directly extracts equations from code [11], but even if we can extract the formula representing the model automatically, the visual feature model is still necessary for human readability.

Despite the initial positive results using ETHOM, we have uncovered some potential limitations that prevent scalability and robustness across a wide range of feature models and problems. First, the feature models derived by ETHOM suffer from low F_1*-measures*. The F_1*-measure* indicates how close the evolved models are to the solution balancing precision and recall. The ETHOM framework includes three fitness functions. The first, FFRelaxed may result in feature models containing a large number of invalid products, while the second fitness FFStrict may go too far in the opposite direction. It uses a penalty for invalid products, but weights this penalty very heavily resulting in models that contain many missed products. The third fitness MinDiff improves upon the FFStrict fitness and improves precision, however it too misses a large number of products.

Second, the ETHOM encoding of cross-tree constraints, supports only simple cross-tree constraints in which both sides of the constraint equation consists of only a single feature, (i.e. *"A requires B"*, or *"B excludes C"*). It cannot hold constraints such as *"A and B implies C"* or *"(A or B) implies (C or D)"*, limiting the types of feature models that can be engineered. We have found in our study that these types of constraints do exist and should be accounted for.

Last, the encoding of a feature diagram does not allow for multiple feature groups that have the same type of relationship with the same parent. A feature group is a set of child features that have a combined-relationship (i.e. OR-group, XOR-group, or MUTEX-group) with their parent feature. The encoding of a feature diagram in ETHOM consists of two parts <PR, CN>, where PR denotes the type of relationship a feature has with its parent, and CN denotes the number of children of the feature. The parent of a feature is implied by its order in the chromosome. Because the encoding does not list the parent feature explicitly, multiple feature groups with the same parent are merged into a single group. Figure 1 shows an example of how this can lead to a modified (incorrect) feature model. The original product line consists of four different XOR-groups (denoted by the arcs between the dependency lines (e.g. V6 or L4 and TWD or AWD). The customer can select either a V6 or L4 engine, TWD or AWD for its drive, W19 or W20 as the wheel size and either a single or dual exhaust system. All of the relationships are merged into a single group in the resulting diagram. Interestingly, the cross-tree constraints for feature L4 and V6 mean that these features can never be in any product, because both of them imply additional features which are now in the same XOR relationship. We have seen this type of relationship in two models from our study (in subjects also used in [13]).

In this paper, we have built upon the work of Lopez-Herrejon et al. [13, 18], and make several improvements to their search-based approach to reverse engineering. We have designed a framework we call *Software Product Line Reverse Engineering Optimization – SPLRevO*. First, we have created a new chromosome representation that allows for more complex cross-tree constraints and that supports multiple group relationships of the same type. Second, we have added new evolutionary operators for crossover and mutation to work on this new structure. Third, we have developed a new fitness function, that penalizes models with invalid products and has higher F-measures than the

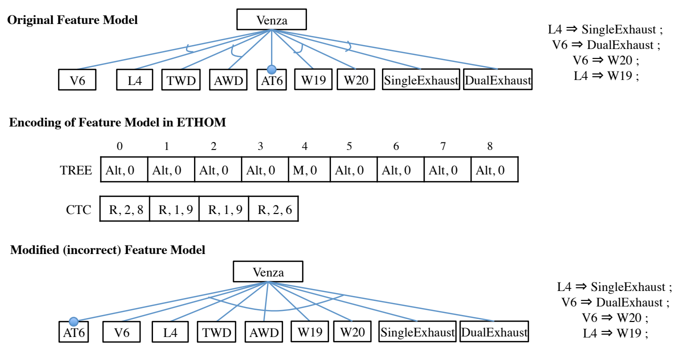


Figure 1. An incorrect encoding for the Toyota Venza feature model

ETHOM fitness functions. Fourth, we show in an empirical study using 99 feature models of various sizes, both with and without complex cross-tree constraints, that our new fitness within the ETHOM framework shows between a 7 and 50% improvement in F-measure over the native ETHOM fitness functions. In addition, when the new fitness is used within SPLRevO, we increase the F- measure over ETHOM from 20 to 63% on average.

The contributions of this work are:

1. A new framework for reverse engineering feature models using a genetic algorithm that supports more complex constraints, multiple feature groups and adds new evolutionary operators as well as incorporates a more accurate fitness function.
2. An empirical study validating our new fitness both within the original ETHOM framework and within our new framework to isolate improvements.

The rest of this paper is organized as follows. The next section provides background and related work. Section 3 presents SPLRevO. In Section 4 we present our empirical study. We discuss our results in Section 5 and end with conclusions, and future work in Section 6.

2. Background and Related Work

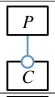
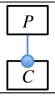
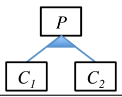
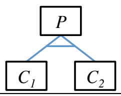
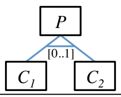
A feature model is a graphical notation describing variability of a software product line consisting of both a feature diagram and a (possibly empty) set of cross-tree constraints. A feature diagram is a tree structure consisting of edges describing the relationships between parent and child features [2]. Every child and parent relationship has a hierarchical edge indicating that the child requires the parent feature. Table 1 shows the most commonly used feature relationships, their graphical notation, and edge dependency constraints. The additional cross-tree constraints that do not have an associated graphical notation are added as a set of logical constraints. We show an example feature model in Figure 2.

In this figure, feature a is a root feature with three children, b, c, and d. Feature b is an optional feature. It has children e and f that are part of a MUTEX-group. When b is selected, either e or f or neither can be selected. Feature c is a mandatory feature. It appears in every product. It has two XOR-groups, g and h, and i and j. This means either g or h, and either i or j must be selected. Feature d is an optional feature. It has children k and l that are members of an OR-group. When d is selected, one or more feature in the group can be selected. The cross-tree constraint (on the right) states that d is always selected when both e and f are selected. Note that although we can describe the set of products in a product line only as equations, this is not enough. More than one feature model may describe the same set of equations, and the graphical view of a feature model is often required for human understandability.

2.1. Existing SPL Analysis Frameworks

ETHOM is the framework that is used for the search-based reverse engineering feature models [13]. ETHOM uses a genetic algorithm. It takes a set of products (all feature sets) as an input. This can be an existing set (from a prior version of the system or reverse engineered from code, or using a tool to analyze an existing feature model, described

Table 1. Notation for Relationships in a Feature Diagram

Relationships	Graphical Notation	Grammar	Feature Edges	Semantic
Optional (hierarchy)		$P : C? ;$	$C \rightarrow P : \mathrm{H}$	When P is selected, C can be selected or deselected.
Mandatory		$P : C ;$	$C \rightarrow P : \mathrm{H}$ $C \rightarrow P : \mathrm{M}$	When P is selected, C must be selected.
OR-group		$P : (C_1\|C_2)+ ;$	$C_1 \rightarrow P : \mathrm{H}$ $C_2 \rightarrow P : \mathrm{H}$ $(C_1, C_2) \rightarrow P : \mathrm{OR}$	When P is selected, either C_1 or C_2 or both of them must be selected.
XOR-group (alternative)		$P : (C_1\|C_2) ;$	$C_1 \rightarrow P : \mathrm{H}$ $C_2 \rightarrow P : \mathrm{H}$ $(C_1, C_2) \rightarrow P : \mathrm{XOR}$	When P is selected, either C_1 or C_2 must be selected.
MUTEX-group		$P : (C_1\|C_2)? ;$	$C_1 \rightarrow P : \mathrm{H}$ $C_2 \rightarrow P : \mathrm{H}$ $(C_1, C_2) \rightarrow P : \mathrm{MUTEX}$	When P is selected, either C_1 or C_2 or none of them must be selected.

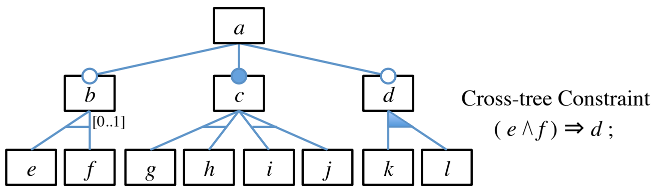


Figure 2. A Feature Model

next). For experimentation, the full set of products can be obtained by providing a set of equations describing the feature model to the FaMa *Feature Model Analyzer* framework [6, 22]. FaMa uses automated reasoners to answer questions about the possible products within a software product line model, such as whether a particular product is a member of the SPL or how many possible products an SPL contains. To generate the initial random population, ETHOM uses a tool called *BEnchmarking and TestTing on the analYses of feature models, (BeTTy)* [14, 23]. BeTTy takes as input the number of products and randomly generates a feature model. During evolution, at each generation, FaMa is again used to evaluate whether the current model matches the set of products. For each product in the target feature set it asks if the current model contains it.

The current version of FaMa (1.1.2) supports two types of feature models, standard (basic) and extended (attributed) feature model – a feature model where features can have additional ranges of values associated with them. The difference between standard and extended FaMa feature model is shown in Table 2. ETHOM only supports the standard model, which limits the framework to simple cross-tree constraints: binary REQUIRES and EXCLUDES. It does not support AND, OR, or NOT directly.

Table 2. FaMa basic and extended models

Features	Standard Model	Extended Model
File extension	*.fm, *.fmf, *.xml	*.afm
%Relationships	Supports Mandatory, Optional, Group Relationships with Cardinality	Supports Mandatory, Optional, Group Relationships with Cardinality
%Constraints	REQUIRES and EXCLUDES	Arithmetic (+, -, *, /, mod) Relational (>, >= , <, <=, ==, !=) Logical (AND, OR, NOT, IMPLIES, IFF (if only if)) operators
%Attributes	Not supported	Defines feature attributes, with their type, domain, and default value

2.2. ETHOM Fitness Functions

ETHOM has three fitness functions, FFRelaxed, FFStrict [13] and MinDiff [18] shown in equations 1, 2 and 3.

$$FFRelaxed(sfs, fm) = |\{fs : sfs \mid validFor(fs, fm)\}| \tag{1}$$

$$FFStrict(sfs, fm) = \begin{cases} FFRelaxed(sfs, fm), & \#products(fm) = |sfs| \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$MinDiff(sfs, fm) = deficit(sfs, fm) + surplus(sfs, fm) \tag{3}$$

In all of the equations *sfs* stands for the set of desired products, *fs* is an individual product in *sfs*, *fm* is a feature model, *validFor*(*fs*,*fm*) is used to check whether *fs* can be described by the feature model *fm* and #*products*(*fm*) counts the number of products in the given feature model. *deficit*(*sfs*,*fm*) is the number of missing products (i.e. the number in *sfs* that are not contained in *fm*). *surplus*(*sfs*,*fm*) is the number of additional products. It counts the number of products in *fm* that are not contained in *sfs*. The first fitness FFRelaxed maximizes the number of products in the evolved feature model that are also in the desired set of products. It ignores invalid products, therefore it may allow for models with many extra (invalid) products. The second fitness FFStrict maximizes the number of products, but does not allow any invalid products. It uses FFRelaxed as its base, but whenever the number of products in the desired feature set and the evolved model do not match, the fitness is set to zero. This penalty, does not provide enough guidance, biasing towards too many missed products. The last fitness, MinDiff , minimizes both the missing and additional products in the feature model, by adding their sums together.

3. SPLRevO

We now present the new chromosome, fitness function and evolutionary operators that form the basis of SPLRevO.

3.1. Chromosome

One aim of SPLRevO is to represent more complex constraints on the feature model. Complex constraints are supported in the FaMa extended feature model so we leverage that to build our chromosome. We use the FaMa extended feature model (without attributes) as the underlying representation. We can then use the FaMa analyzer to answer questions about our model as during evolution[1]. Table 3 shows the operators for arbitrary constraints that we support in comparison with those supported by the ETHOM chromosome.

Table 3. Supported operators for constraints in ETHOM and SPLRevO

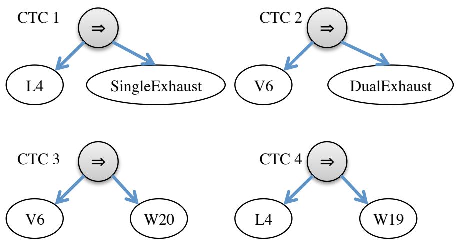
Framework	Supported Operators
ETHOM	REQUIRES, EXCLUDES
SPLRevO	REQUIRES, EXCLUDES, AND, OR, NOT, IMPLIES, IFF

The SPLRevO chromosome is split into two parts, the feature diagram, and the cross-tree constraints. The feature diagram part of the chromosome contains one or more children, the parent, and a relationship. The relationship for a single child can be a Hierarchy and Mandatory, MUTEX, OR, and XOR for a group of children. The cross-tree constraints are represented by their expression tree. The encoding of the Toyota Venza feature model from Figure 2 is shown in Figure 3. Part (a) is the feature diagram and part (b) is the set of cross-tree constraints in tree form (they are flattened into an array form in our implementation). In Figure 3, the feature diagram now explicitly encodes the parent feature of each feature and feature group. Thus, we solve the problem of merging feature groups. In addition, we encode the CTC in tree form which allows for arbitrary sized constraints with more than one operator. This solves the ETHOM limitation of only allowing simple binary constraints.

[1]FaMa does not support some questions using the attributed model, but the technical limitation is due to the attributes. We instead query the solver directly to support SPLRevO.

	Children	Parent	Type
R1	V6	Venza	Hierarchy
R2	L4	Venza	Hierarchy
R3	{V6, L4}	Venza	Xor
R4	TWD	Venza	Hierarchy
R5	AWD	Venza	Hierarchy
R6	{TWD, AWD}	Venza	Xor
R7	AT6	Venza	Hierarchy
R8	AT6	Venza	Mandatory
R9	W19	Venza	Hierarchy
R10	W20	Venza	Hierarchy
R11	{W19, W20}	Venza	Xor
R12	SingleExhaust	Venza	Hierarchy
R13	DualExhaust	Venza	Hierarchy
R14	{SingleExhaust, DualExhaust}	Venza	Xor

(a) Feature Diagram



(b) Cross-tree Constraints

Figure 3. Encoding of Feature Model for Toyota Venza Feature Model: (a) diagram (b) CTCs

3.2. *Fitness: FFValidity*

We include a new fitness function FFValidity in SPLRevO. It is designed to maximize coverage of the desired products while minimizing the number of undesired products. The MinDiff fitness adds the deficit to the surplus products and then minimizes the fitness. Our fitness on the other hand, subtracts the additional products from the matched products, and normalizes this by the number of desired products. It also uses a logarithmic scale to smooth out the fitness curve. Our fitness, FFValidity, is shown in equations 4 and 5.

$$FFValidity(sfs, fm) = \frac{(\log_2(\#matched(sfs, fm) + 1) - 0.1 \cdot \log_2(\#additional(sfs, fm) + 1)) \cdot 100}{\log_2(|sfs| + 1)} \tag{4}$$

$$\#additional(sfs, fm) = \#products(fm) - \#matched(sfs, fm) \tag{5}$$

In equation 4, $\#matched(sfs, fm)$ is the number of the desired products that are described by the generated feature model fm. The calculation of $\#matched(sfs, fm)$ is equivalent to $FFRelaxed(sfs, fm)$ but we call it matched which is more meaningful to us. $\#additional(sfs, fm)$ is the number of invalid products that are described by fm. The term $\log_2(\#additional(sfs, fm) + 1)$ is the penalty applied when fm describes extra products. We use $\log_2$ to reduce the effect caused by the large number of products which is usually around $2^{\#features}$. We heuristically weight the penalty at 10% to lessen its impact on the matched products: $\log_2(\#matched(sfs, fm) + 1)$. We also add 1 inside of our $\log_2$ formulas to ensure that they go to zero. Finally, we normalize the value by $\frac{100}{\log_2(|sfs|+1)}$. The final fitness has a range from 0 to 100; 100 is the correct model. The idea of this fitness is to drive the search towards more matched products, while only slightly penalizing additional products. It gives higher priority to the number of matched products. Thus, the genetic algorithm will first converge towards covering all of the products when possible, and gradually refine towards models that contain fewer additional products.

3.3. *Crossover and Mutation*

Given the more complex chromosome representation we needed to add additional crossover and mutation operators. We describe them here. For each pair of parents to be mated, we crossover the feature diagram and cross-tree constraints separately, and generate two new offspring. Our new crossover operator on the feature diagram is able to transfer relationships under a random feature f, considered as a root of the subtree of the diagram, from one parent to another parent without distorting the relationships in the tree structure, which may happen in the ETHOM crossover. We use a simple even-odd pairing of chromosomes from the top half of the population. Thus, the total number of chromosomes after crossover is the same as the initial population. The algorithms used for crossover are shown as algorithms 1 - 3.

Algorithm 1: crossover(P_1, P_2)

```
Result: Two new offspring
Input: Two parents, $P_1$ and $P_2$
1 $f_1 = random(features)$;
2 $offspring_1 = crossoverFD(P_1, P_2, f_1)$;
3 $f_2 = random(features)$;
4 $offspring_2 = crossoverFD(P_2, P_1, f_2)$;
5 $crossoverCTC(offspring_1, offspring_2)$;
```

Algorithm 2: crossoverFD(P_2, P_1, f)

```
Result: A new offspring
Input: A source parent $P_1$, a target parent $P_2$, and a random feature $f$
1  $Offspring = P_2.clone()$;
2  $enqueue(F, f)$;
3  while $F \neq \emptyset$ do
4      $f' = dequeue(F)$;
5      for each hierarchical relationship $P_1.r$ from $f'$ to each child $P_1.c$ on $P_1$ do
6          $enqueue(F, P_1.c)$;
7          disconnect $c$ from its parent $p$ ;                                  // Figure 4(2)
8          if $c \in predecessor(f')$ then
9              disconnect $c'$, $c' \in child(c)$ and $c' \in predecessor(f')$ ;   // Figure 4(3)
10             reconnect $c'$ to $p$ using relationship $P_2.c \rightarrow p$ ;     // Figure 4(4)
11         connect $c$ to $f'$ using relationship $P_1.r$ ;                        // Figure 4(5)
12 return $Offspring$
```

Algorithm 3: crossoverCTC($offspring_1$, $offspring_2$)

```
Result: Exchanged cross-tree constraints in two offspring
Input: Two offspring, $offspring_1$ and $offspring_2$
1 $M = Integer.random()$;
2 $N = Integer.random()$;
3 $ctcO_1 = getCTC(offspring_1, M)$;
4 $ctcO_2 = getCTC(offspring_2, N)$;
5 $offspring_1.addAllConstraints(ctcO_2)$;
6 $offspring_2.removeAllConstraints(ctcO_2)$;
7 $offspring_2.addAllConstraints(ctcO_1)$;
8 $offspring_1.removeAllConstraints(ctcO_1)$;
```

In Algorithm 1, crossover(P_1, P_2) is the main procedure for crossover between two parent chromosomes, P_1, and P_2. We select two random features (lines #1 and #3) and then clone the relationships under those features to the respective parent, as shown in Algorithm 2. Then we exchange constraints between two new offspring. This is shown as Algorithm 3. To perform crossover on the individual parent, (Algorithm 2) a new offspring is first created from a copy of another parent P_2. Then, the relationships under the feature f are transferred to the new offspring. We iteratively transfer the relationships of all features in the subtree starting from its root feature, f. For each relationship r from an active feature f' to c, under the feature f on the source parent, P_1, form r on the offspring. In the offspring, we disconnect the child c of the relationship r from its parent feature p. This step is to prepare c for connecting to a new parent, f'. However, If c is a predecessor of f', we cannot connect c to f' directly. This is because it will cause a cycle in the tree, which is an invalid feature model. In this case, we have to perform two extra steps (lines #9 and #10). If c is a predecessor of f', disconnect a child c' from c. Then, reconnect c' to the parent p using the same relationships $P_2.c \rightarrow p$. Otherwise, when c is not or no longer be a predecessor of f', connect c to f' using the relationship r. This crossover operation is illustrated in Figure 4.

crossoverCTC($offspring_1$, $offspring_2$) (Algorithm 2) is the core process of crossover for cross-tree constraints for two new offspring originally created from P_1 and P_2. This process exchanges the first M and N constraints from the two parents to the two new offspring, where M and N are random integers.

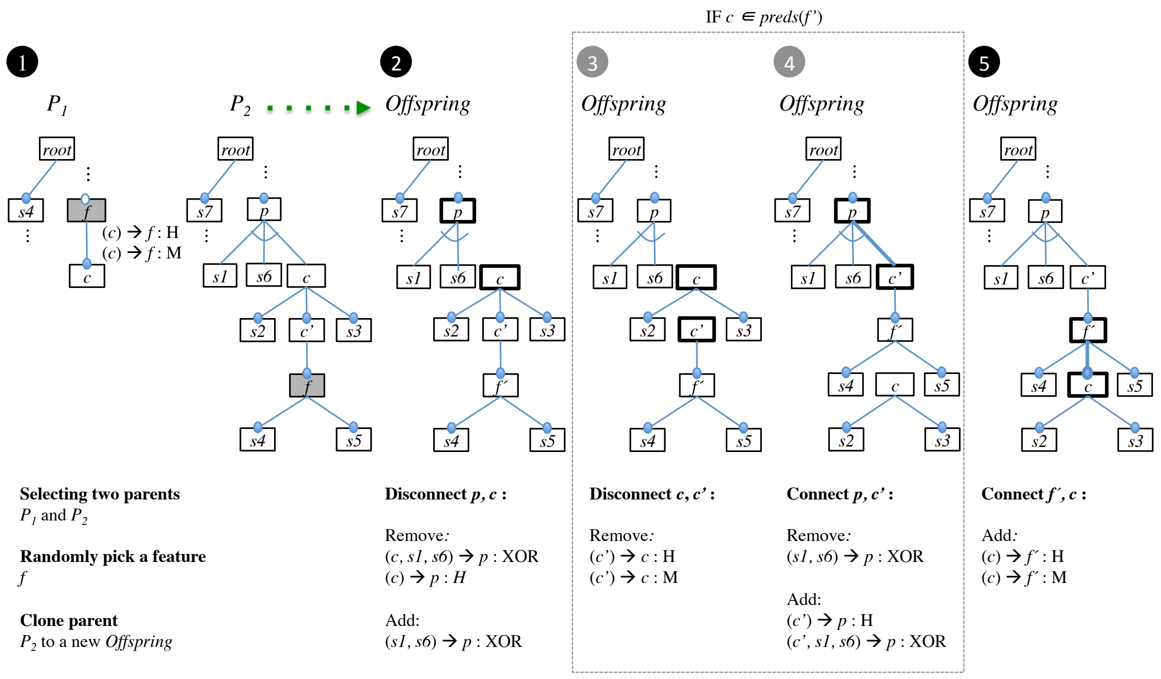


Figure 4. An example of crossover of the feature diagram between parent P_1 and P_2

3.4. Mutation

We have created ten new mutation operators, five for the feature diagram and five for the cross-tree constraint. Our mutation avoids infeasible feature models, therefore we do not need a repair mechanism. We describe each next.

FD Complicate. Figure 5 shows an example of this operator. It adds more features to the children of an existing group relationship, or makes a new group relationship. It randomly picks a feature f to be added as a new child to a random relationship r. To avoid a cyclic graph, f must not be a predecessor of any feature in r. In the case that r is a supporting hierarchical edge of a group relationship g, resume the operation on g instead of r. This avoids multiple feature cardinality, where one feature relies on more than one group relationship. If r is a non-group relationship, randomly change the type of r to OR, XOR, or MUTEX after adding the feature f. This operator can be applied to any relationship in the feature diagram. An example of making the relationship $F7 \to F2$: M more complicated by adding a feature $F4$ is shown in Figure 5.

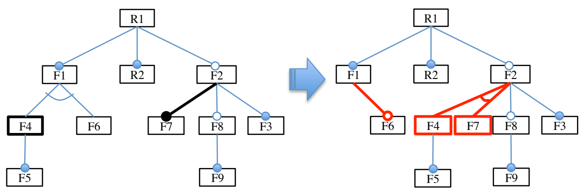


Figure 5. Feature Diagram Mutation: Complicate

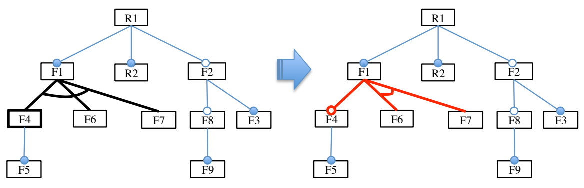


Figure 6. Feature Diagram Mutation: Simplify

FD Simplify. This operator (Figure 6) makes the feature diagram simpler by removing a child feature f from an existing group relationship r. If removing f from r results in only one child, the group relationship r can be removed. This operator can be applied only to a group relationship in the feature diagram. The example of this operator that removes $F4$ from a group relationship $(F4, F6, F7) \to F1$: XOR is shown in Figure 6. It first randomly picks a group relationship, $(F4, F6, F7) \to F1$: XOR, from the feature diagram. There is only one group relationship in this case. Then, it randomly picks a child feature, $F4$, to be removed from the relationship.

FD Swap Features. This operator swaps two features in the feature diagram. It allows exchange of the relationships of the two features. An example of this operator on $F2$ and $F4$ is shown in Figure 7.

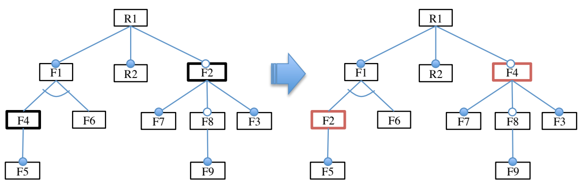


Figure 7. Feature Diagram Mutation: Swap Features

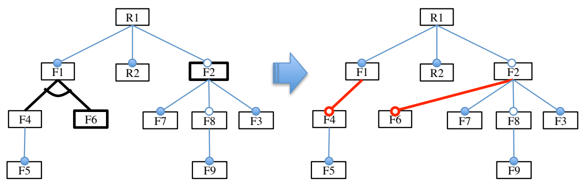


Figure 8. Feature Diagram Mutation: Change Parent

FD Change Parent. This operator changes the parent of a feature. To avoid a cyclic graph, a feature cannot change its parent to its successor. An example that changes the parent of $F6$ from $F1$ to $F2$ is shown in Figure 8.
FD Change Type of Relationship. This operator changes the type of any relationship in the feature diagram. If it is a hierarchical relationship and not related to any group relationship, add a mandatory relationship. If it is mandatory, remove the relationship to make optional. If it is a group relationship, randomly change to the other group relationship. An example of this operator that changes an XOR-group relationship to an OR-group relationship is shown in Figure 9.
CTC Expand. This operator makes a cross-tree constraint more complex by expanding a feature node in an expression tree to a new expression subtree. This operation can be applied only to a feature node that is a leaf node of the expression tree. Figure 10(a) shows an expression tree of a constraint $(F5 \wedge F3) \Rightarrow (\neg F8 \vee F9)$. Expanding a feature node $F8$ to a new expression subtree is shown in Figure 10(b). A feature node $F8$ is expanded to $F8\langle op\rangle R$, where $\langle op\rangle$ is one of the available logical operators $(\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow)$, where R is a random feature.
CTC Collapse. This operator makes a cross-tree constraint simpler by collapsing a leaf expression tree to a single feature node. This operator can be applied only to an expression node where all its children are feature nodes. An example of this operator that collapses an expression $F5 \wedge F3$ to a single feature node $F3$ is shown in Figure 12(c). An expression node $\wedge$, having two children feature nodes, $F5$ and $F3$, is collapsed to only $F3$. $F3$ is a random child feature node of the original expression.
CTC Change. This operator changes information in a node in an expression tree. If the node is a feature node, its information can be changed to another feature. Otherwise, if it is an expression node, its operator can be changed to another operator. The example of an operator that changes a conjunctive operator to disjunctive operator is shown in Figure 10(d). The expression node $\wedge$ is randomly changed to another operator $\vee$.
The last two operators that we include are described, but not illustrated:
CTC Add New Constraint. This operator adds new constraint to the feature model. The new constraint is a binary expression of two features and a random operator. We do not check whether the new constraint is already satisfied by the feature diagram or it is a duplicate constraint.
CTC Remove Constraint. The last operator removes an existing constraint from the feature model.

3.5. Mutation Rates

Since the mutation operators make significant changes to a chromosome and the population, SPLRevO is configurable with respect to the mutations. Based on the overall mutation rate, we also provide a rate to use for individual

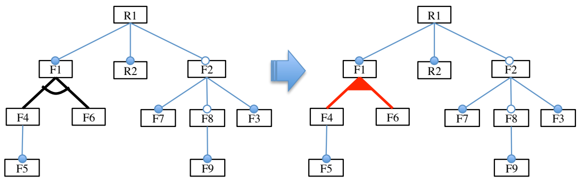


Figure 9. Feature Diagram Mutation: Change Type of Relationship

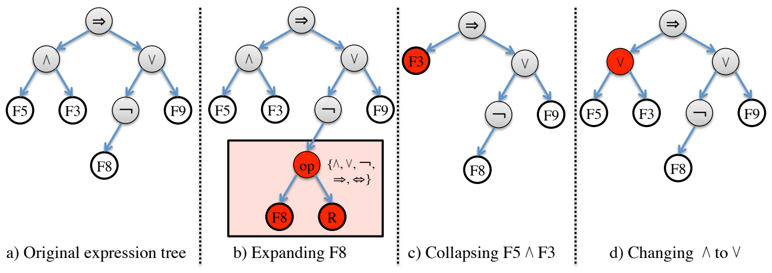


Figure 10. Cross-tree Constraint Mutation Operators

operators (we have found that if we use all operators equally, then some dominate others). We call this the *applicable rate* or A-Rate for short. It is automatically and dynamically calculated during evolution, based on the current population. The list of mutation operators and their applicable rates (A-Rate) are shown in Table 4.

Table 4. Mutation operators and dynamic applicable rate

Operator	**Mutation Point**	**A-Rate**
FD::Complicate	Any relationship	1.0
FD::Simplify	Any group-relationship	$^{\lvert grp-relationships\rvert}/_{\lvert all-relationships\rvert}$
FD::SwapFeature	Any relationship	1.0
FD::ChangeParent	Any relationship	1.0
FD::ChangeRtype	Any relationship	1.0
CTC::Expand	Any leaf node	$^{\lvert leaf-nodes\rvert}/_{\lvert all-nodes\rvert}$
CTC::Collapse	Any non-root node which all its children are leaf nodes	$^{\lvert collapsible-nodes\rvert}/_{\lvert all-nodes\rvert}$
CTC::Change	Any non-root node	$^{\lvert all-nodes\rvert-1}/_{\lvert all-nodes\rvert}$
CTC::Add	Any chromosome in the population	1.0
CTC::Remove	Any constraint in the population	1.0

3.6. Other GA Parameters

The SPLRevO initial population is generated randomly. These are models without CTCs. Then we randomly mutate the population using the set of mutation operators to add some constraints. For selection we use rank selection. We first order the chromosomes in the population by their fitness value. We keep the best half of the population for crossover and the next generation, while discarding the later half. The stopping criteria is either a model that matches the desired set of products (with no additional or missing products) or an iteration limit.

4. Empirical Study

In this section, we evaluate the three ETHOM fitness functions, along with FFValidity, on the ETHOM framework and on the SPLRevO framework. We aim to answer the following research questions.
RQ1: How does FFValidity perform compared to the ETHOM fitness functions?
RQ2: What is the impact of the new chromosome representation and genetic operators in SPLRevO?

Table 5. Feature Models Used in Experiments

Model	**Total Models**	**#Features-range**	**Avg. #Features**	**Avg. #Products**
Small	21	10	10.0	22.1
Medium	55	11 - 19	14.0	102.9
Large	15	20 - 27	21.9	227.9
Complex	8	7 - 21	13.3	131.6
Total	99	10 - 27	14.3	107.0

4.1. Feature Models

We used 99 feature models for our experimentation. 91 of these models were used in the original ETHOM study, although the authors only reported on 59 [13]. We used all models with 27 or fewer features. We chose this cutoff so that the results are reasonable to complete during the timeframe. These models are originally from the SPLOT feature model repository [24]. We separate these into three groups, small, medium, and large, based on the number of features. We also include 8 models that have complex cross-tree constraints. First we used the Power Management feature model from the work of She et al. [5]. We also included our example Toyota Venza model from Figure 2. For the other six we randomly choose two models from each group (small, medium and large) and mutated them randomly so that they contain complex constraints. We manually verified that they contained a complex constraint pattern. Table 5 shows the details of our models including the number of features, and the average number of products. A more in-depth description of the complex models is given in Table 6.

Table 6. Complex subjects

Model	#Features	#Products	#CTC	#CCTC	Pattern of complexity
Power Management [5]	7	6	4	1	(A AND B) IMPLIES C
Toyota Venza M	10	4	4	0	Multiple groups (4 XOR-groups)
carpl M	10	21	0	0	Multiple groups (1 XOR-group, 1 MUTEX-group)
CTOS MODEL M	10	17	1	1	(A AND B) IMPLIES (C OR D)
CFDP Library M	13	549	1	1	(A AND B) IMPLIES C
Web Content Delivery M	15	66	5	1	A IMPLIES (B OR C)
Stacja Pogodowa M	20	336	1	1	(A IFF B) IMPLIES C
TrialADC M	22	54	5	1	(NOT A) IMPLIES B

4.2. Metrics Used

We evaluate the effectiveness of the fitness functions by comparing the desired set of products with the products in the evolved feature model. We count the missing and additional products in each model and calculate the percent of each. We also calculate the *precision*, *recall*, and the F_β*-measure* as measures of effectiveness. The F_β *measure* is the weighted harmonic mean of precision and recall. In this paper, we use F_1, where $\beta = 1$, that weights precision and recall equally [25]. A high F_1-measure is our primary goal since this indicates we have both high precision and high recall. The calculations for our metrics are given in equations 6 - 10. In these equations *sfs* is the number of desired products, *fm* is the evolved feature model (the one we are evaluating). We also record the execution time that each experiment takes to measure efficiency.

$$precision(sfs,fm) = \frac{\#matched(sfs,fm)}{\#products(fm)} \cdot 100 \tag{6}$$

$$recall(sfs,fm) = \frac{\#matched(sfs,fm)}{|sfs|} \cdot 100 \tag{7}$$

$$F_{\beta=1}(sfs,fm) = \frac{2 \cdot precision(sfs,fm) \cdot recall(sfs,fm)}{precision(sfs,fm) + recall(sfs,fm)} \tag{8}$$

$$\%Missing(sfs,fm) = \frac{|sfs| - \#matched(sfs,fm)}{|sfs|} \cdot 100 \tag{9}$$

$$\%Additional(sfs,fm) = \frac{\#products(fm) - \#matched(sfs,fm)}{|sfs|} \cdot 100 \tag{10}$$

4.3. Method

We run all experiments 10 times and then present averages (and medians using boxplots). For all subjects, and fitness functions we run them in both ETHOM and in SPLRevO. We run all experiments on the same computing cluster with AMD Opteron(TM) CPUs running at 2300MHz. Each individual configuration was submitted to a separate node with a maximum Java memory pool of 16GB. The genetic algorithm configurations that we used to run ETHOM and SPLRevO in our studies are shown in Table 7.

Table 7. ETHOM and SPLRevO configuration

Parameter	ETHOM	SPLRevO
Fitness functions	FFRelaxed, FFStrict, MinDiff, FFValidity	FFRelaxed, FFStrict, MinDiff, FFValidity
Selection strategy	Roulette-wheel	Rank Selection
Crossover strategy	One-point	One-point
Crossover probability	0.7	1.0
Initial population size	100	100
Infeasible individuals	Repairing	Avoidance
Maximum generations	100	100
Maximum iterations	25	25
Mutation probability	0.01	0.01
Percentage of CTC	0.3	N/A

4.4. *Threats to Validity*

This study suffers from some threats to validity which we outline here. First, with respect to external threats (or generalization), we used only models from the SPLOT repository and these have no more than 27 features, therefore we cannot be sure that our algorithm will scale to very large feature models. However, these are the same subjects used in the original work and they are meant to represent a broad range of real product lines. We have also added some models with complex constraints to enrich the data set. As for internal threats to validity, we cannot be sure that there are no errors in our analysis, however we have manually verified many of the experiments to confirm that they are correct. With respect to the comparison of the fitness functions (RQ1), we isolated our function within the original ETHOM framework to avoid external noise (such as different parameter tuning or efficiency of data structures). We did the same within the SPLRevO framework. There may, however, be some effect from these differences in RQ2 where we cannot completely isolate the new representation and operators from other implementation differences. Finally, with respect to construct validity, we may have chosen different metrics, however, the ones we use, additional products, missing products and the F_1*-measure* are all used in the prior studies.

5. Results

We now answer each of our research questions in turn. To answer RQ1 (quality of FFValidity) we turn to Table 8 and to Figures 11 to 16 . In Table 8, we show results both from the ETHOM and from the SPLRevO framework for the average of 10 iterations on 99 feature models. For each framework, the best value within each category is shown in bold font. Italic font shows a positive improvement on SPLRevO over ETHOM. The cells with borders are the best values found across both frameworks. The boxplots in Figures 11 to 16 plot the individual results of 10 iterations on 99 feature models. Figures 11 to 14 show the F_1*-measure* in each category (described in Table 5). Figure 15 and 16 show %Missing and %Additional for all feature models, respectively.

For the first RQ, we look primarily at the results from ETHOM (where we isolate the new fitness using the old chromosome and genetic operators). In Table 8, we can see that with respect to missing products, FFValidity has the lowest rate (no higher than 15.4 percent). The next best fitness with respect to missing products is the FFRelaxed fitness function, but as we shall see this suffers from too many additional products. The FFStrict and the MinDiff are missing as many as 95% and 73% respectively of the required products. If we turn to additional products we see that the MinDiff fitness performs the best, however, FFValidity is better than FFRelaxed and FFStrict fitness functions (except in the complex models). The FFRelaxed performs the worst as expected since it does not penalize for this metric. When we turn to precision, we see that MinDiff again performs slightly better than FFValidity but both of the other fitness functions are much worse. With respect to recall, we see that FFValidity is the best, followed next by FFRelaxed. Finally, we turn to the F_1*-measure* which is the combined measure. We see that with respect to the F_1*-measure*, FFValidity performs the best with an average between 37.7% (in complex) to 70.5% in the small models. The next best fitness is MinDiff which is slightly below this. The trend seen when looking at the medians of the boxplots (Figures 11 to 14) holds. We also look at the run time in minutes for each fitness. FFValidity is the most costly algorithm time wise, but takes on average no more than 10 minutes longer to complete than the others (within ETHOM).

We next ran the Mann-Whitney-Wilcoxon Test (in R) to determine if these results are significant. We compare the F_1*-measure* between FFValidity and each of the three existing ETHOM fitness functions, using a 0.05 significance level. The null hypothesis that there is no significance improvement in the F_1*-measure* of FFValidity over each of the existing ETHOM fitness functions, FFRelaxed, FFStrict, and MinDiff. The result for this analysis is shown in Table 9. We show the p-value. We can conclude that FFValidity shows a significant improvement overall in the F_1*-measure* over FFRelaxed and FFStrict. However, when focusing on particular groups of models, we see that FFValidity does not show a significant improvement for the small and complex models over MinDiff.

Table 8. The quality of fitness functions on SPLRevO and ETHOM separated by size and complexity of feature models

		% Missing					% Additional				
Framework	**Fitness**	**Small**	**Medium**	**Large**	**Complex**	**Average**	**Small**	**Medium**	**Large**	**Complex**	**Average**
ETHOM	FFRelaxed	10.0	7.8	9.9	16.6	9.3	1396.8	8003.8	187667.0	25160.8	35210.4
	FFStrict	63.7	80.5	95.0	85.3	79.5	453.6	604.1	48883.8	302.8	7862.9
	MinDiff	38.7	50.5	55.4	73.1	50.6	**8.0**	**7.2**	**11.3**	**9.4**	**8.2**
	FFValidity	**8.7**	**7.1**	**9.4**	**15.4**	**8.5**	115.1	334.8	3086.8	1558.2	804.1
SPLRevO	FFRelaxed	***0.0***	***0.1***	***0.3***	***0.4***	***0.1***	2761.8	32361.5	3181742.8	86795.4	507660.4
	FFStrict	64.7	80.8	95.1	85.4	80.0	858.4	5229.0	729394.5	8817.2	114313.9
	MinDiff	56.4	82.5	99.5	82.8	79.6	***2.4***	***0.7***	***0.0***	***1.0***	***1.0***
	FFValidity	*0.7*	*1.0*	*6.1*	*1.4*	*1.7*	*52.3*	*114.9*	*3855.6*	*566.7*	*704.9*

		Precision (%)					Recall (%)				
Framework	**Fitness**	**Small**	**Medium**	**Large**	**Complex**	**Average**	**Small**	**Medium**	**Large**	**Complex**	**Average**
ETHOM	FFRelaxed	12.6	7.1	1.0	7.3	7.3	90.0	92.2	90.1	83.4	90.7
	FFStrict	33.7	18.6	1.8	15.7	19.0	36.3	19.5	5.0	14.7	20.5
	MinDiff	**84.7**	**78.8**	**69.6**	**63.5**	**77.4**	61.3	49.5	44.6	26.9	49.4
	FFValidity	65.1	57.4	50.5	30.1	55.8	**91.3**	**92.9**	**90.6**	**84.6**	**91.5**
SPLRevO	FFRelaxed	7.4	3.9	0.1	5.5	4.2	***100.0***	***99.9***	***99.7***	***99.6***	***99.9***
	FFStrict	31.8	13.6	0.2	11.4	15.3	35.3	19.2	4.9	14.6	20.0
	MinDiff	***95.4***	***96.6***	***100.0***	***96.5***	***96.1***	43.6	17.5	0.5	17.2	20.4
	FFValidity	*76.5*	*71.1*	*53.1*	*49.7*	*67.9*	*99.3*	*99.0*	*93.9*	*98.6*	*98.3*

		Time (min)					F_1*-measure*				
Framework	**Fitness**	**Small**	**Medium**	**Large**	**Complex**	**Average**	**Small**	**Medium**	**Large**	**Complex**	**Average**
ETHOM	FFRelaxed	**0.8**	**6.1**	**44.7**	**12.2**	**11.3**	20.1	11.9	1.8	12.1	12.1
	FFStrict	4.2	17.9	57.1	22.2	21.3	32.7	18.0	1.9	14.7	18.4
	MinDiff	4.6	28.1	128.7	37.9	39.1	67.9	54.9	47.0	34.2	54.8
	FFValidity	4.9	33.2	143.0	45.7	44.8	**70.5**	**64.3**	**56.2**	**37.7**	**62.3**
SPLRevO	FFRelaxed	***0.1***	***2.7***	***17.6***	**12.7**	***5.2***	13.1	6.9	0.3	9.7	7.4
	FFStrict	*1.8*	*7.5*	*30.5*	*15.1*	*10.4*	31.0	13.8	0.2	11.1	15.1
	MinDiff	*1.9*	*10.2*	*31.6*	*13.7*	*12.0*	49.7	20.1	0.6	23.8	23.7
	FFValidity	*2.3*	*21.5*	815.2	66.5	141.3	***84.0***	***78.9***	***55.4***	***57.5***	***74.7***

Table 9. p-value of statistical analysis for RQ1 : comparing F_1*-measure* between FFValidity and three existing ETHOM fitness functions within ETHOM using Mann-Whitney-Wilcoxon Test

Fitness	**Small**	**Medium**	**Large**	**Complex**	**Overall**
FFRelaxed	< 2.20E-16	< 2.20E-16	< 2.20E-16	1.26E-10	< 2.20E-16
FFStrict	< 2.20E-16	< 2.20E-16	< 2.20E-16	1.55E-09	< 2.20E-16
MinDiff	**0.2790**	2.10E-06	0.0145	**0.2128**	1.02E-06

To answer RQ2 we examine the fitness functions within the SPLRevO framework. If we examine Table 8, we see that two of the ETHOM fitness functions FFRelaxed and FFStrict do not perform as well as they did in ETHOM, except with respect to time. MinDiff, on the other hand, which was the best of the ETHOM fitness functions improves the most with respect to precision within SPLRevO, but seems to suffer with recall and the overall F_1*-measure*. FFRelaxed has the best recall (99.9%) as it covers almost all the desired products. It also runs the fastest requiring no

Table 10. p-value of statistical analysis for RQ2 : comparing F_1*-measure* between ETHOM and SPLRevO on each fitness functions using Mann-Whitney-Wilcoxon Test

Fitness	Small	Medium	Large	Complex	Overall
FFRelaxed	3.11E-07	< 2.20E-16	< 2.20E-16	**0.2955**	< 2.20E-16
FFStrict	**0.6072**	< 2.20E-16	9.55E-05	0.0083	< 2.20E-16
MinDiff	2.57E-06	< 2.20E-16	< 2.20E-16	0.0005	< 2.20E-16
FFValidity	5.96E-06	2.28E-14	**0.7443**	3.07E-05	< 2.20E-16

more than 6 minutes. MinDiff is the best for precision (96.1%) as it limits the number of invalid products. FFValidity is the second best in precision, recall, %Missing and %Additional, but it is the best for the overall F_1*-measure* (74.7%). An interesting phenomenon is that the best average value for each of the metrics is found within SPLRevO, as shown in the distinct border in Table 8.

We computed the number of complex constraints in our resulting models to understand if these would persist during evolution. We found that both the complex propositional formula, plus the multiple group phenomenon is seen in our data, although this is a small percentage of the models. For instance, we see on average no more than two complex constraints per model and only two models have the multiple group relationship. But these are non-zero, therefore we do believe they are necessary to support.

Given these results we can conclude that the combination of using FFValidity within the SPLRevO framework provides the highest F_1-measures for our models. Using our new operators and chromosome seems to also improve the F_1*-measure*. We do indeed have complex constraints in our models and see some grouped feature relationships that are not supported by the ETHOM framework. However, we witnessed a decrease in efficiency (SPLRevO using FFValidity); it requires more computation time. We plan to evaluate this tradeoff as future work.

We next ran the Mann-Whitney-Wilcoxon Test to compare the F_1*-measure* of SPLRevO versus ETHOM on each fitness function. Again we use a 0.05 significance level. The null hypothesis for RQ2 is that there is no significance improvement in the F_1*-measure* of SPLRevO over ETHOM. The result for this analysis is shown in Table 10. We conclude that for the overall data, we can reject the null hypothesis for all four fitness functions, however SPLRevO only makes a significance improvement over ETHOM for FFValidity, while the other fitness functions are instead significantly worse. We can see some exceptions to the null hypothesis rejection, when we examine individual groups.

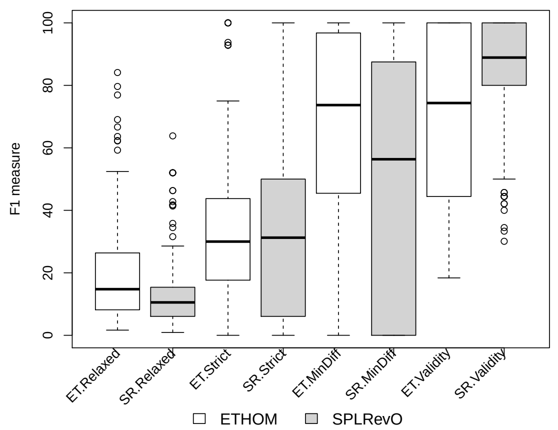


Figure 11. F_1*-measure* of Small Models

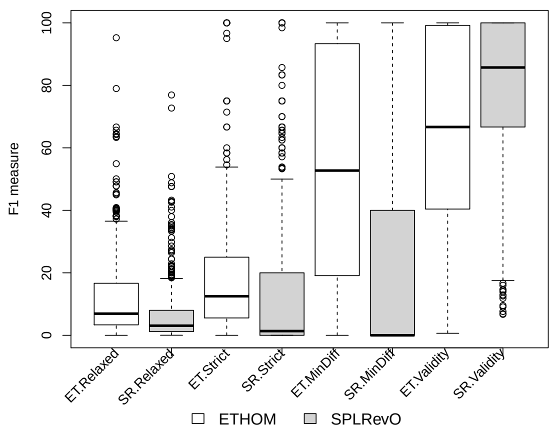


Figure 12. F_1*-measure* of Medium Models

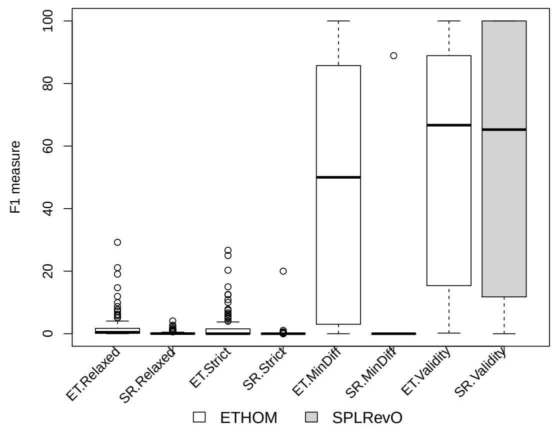


Figure 13. F_1-*measure* of Large Models

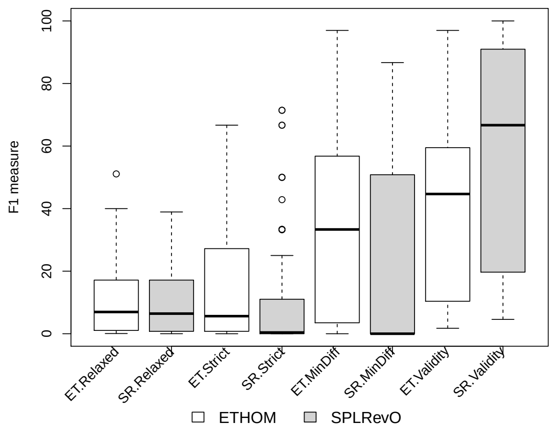


Figure 14. F_1-*measure* of Complex Models

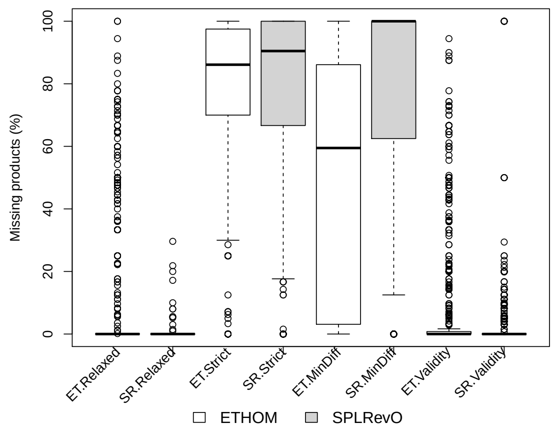


Figure 15. %Missing of All Models

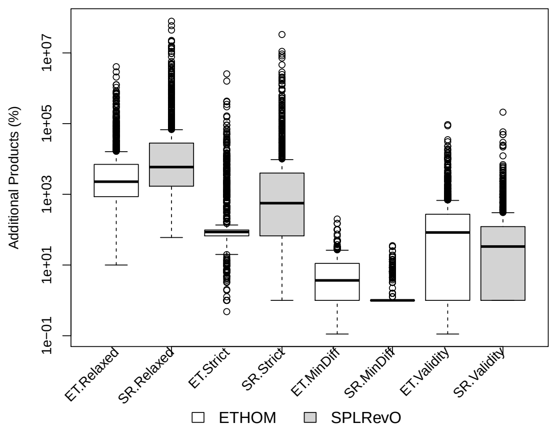


Figure 16. %Additional of All Models

6. Conclusions and Future Work

In this paper, we presented a new framework, SPLRevO for reverse engineering feature models. We proposed a new fitness function called FFValidity and a new chromosome representation and evolutionary operators. We have experimented with 99 subjects on both the ETHOM and SPLRevO framework on a wide range of model sizes and cross-tree constraint complexity. The results show that FFRelaxed is the fastest fitness function but has the lowest F_1-measure. MinDiff has the highest precision, but suffers from more missed products than FFValidity which has the highest F_1-measures When the ETHOM fitness functions are run within SPLRevO, two of them perform slightly worse, however the best overall results for each metric was found using SPLRevO. We also see that there are complex constraints and relationships in the models examined, therefore we believe that this new framework is needed. In future work we will perform additional experiments and evaluate the time/effectiveness tradeoff of these algorithms. We will also look more at complex feature models to understand the occurrence of this phenomenon in real SPLs.

Acknowledgments

This research was supported in part by the National Science Foundation awards CCF-1161767 and CNS-1205472.

References

[1] P. Clements, L. M. Northrop, Software Product Lines: Practices and Patterns, Addison Wesley, 2001.

[2] D. Batory, Feature models, grammars, and propositional formulas, in: International Conference on Software Product Lines (SPLC), 2005, pp. 7–20.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature–oriented domain analysis (foda) feasibility study, Tech. Rep. CMU/SEI-90-TR-21 (nov 1990).

[4] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering, Springer, Berlin, 2005.

[5] S. She, R. Lotufo, T. Berger, A. Wąsowski, K. Czarnecki, Reverse engineering feature models, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011, pp. 461–470.

[6] D. Benavides, S. Segura, P. Trinidad, A. Ruiz-cortés, FaMa: Tooling a framework for the automated analysis of feature models, in: In Proceeding of the First International Workshop on Variability Modelling of Softwareintensive Systems (VAMOS, 2007, pp. 129–134.

[7] C. H. P. Kim, D. S. Batory, S. Khurshid, Reducing combinatorics in testing product lines, in: International Conference on Aspect-Oriented Software Development, 2011, pp. 57–68.

[8] S. Segura, R. Hierons, D. Benavides, A. Ruiz-CorteÌĄs, Automated test data generation on the analyses of feature models: A metamorphic testing approach, in: Software Testing, Verification and Validation (ICST), 2010 Third International Conference on, 2010, pp. 35–44.

[9] I. Cabral, M. B. Cohen, G. Rothermel, Improving the testing and testability of software product lines, in: International Conference on Software Product Lines, 2010, pp. 241–255.

[10] S. Reis, A. Metzger, , K. Pohl, Integration testing in software product line engineering: A model-based technique, in: Fundamental Approaches to Software Engineering, 2007, pp. 321–335.

[11] S. Nadi, T. Berger, C. Kästner, K. Czarnecki, Mining configuration constraints: Static analyses and empirical results, in: Proceedings of the 36th International Conference on Software Engineering (ICSE), 2014, pp. 140–151.

[12] O. Greevy, Enriching reverse engineering with feature analysis, Ph.D. thesis, Univeristy of Bern, Bern, Switzerland (April 2007).

[13] R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, A. Egyed, Reverse engineering feature models with evolutionary algorithms: An exploratory study, in: Symposium on Search Based Software Engineering, Vol. 7515 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 168–182.

[14] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, A. Ruiz-Cortés, BeTTy: Benchmarking and testing on the automated analysis of feature models, in: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '12, 2012, pp. 63–71.

[15] C. Kästner, A. Dreiling, K. Ostermann, Variability mining: Consistent semiautomatic detection of product-line features, IEEE Transactions on Software Engineering 40 (1) (2014) 67–82.

[16] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, P. Lahire, On extracting feature models from product descriptions, in: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '12, 2012, pp. 45–54.

[17] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, P. Heymans, Feature model extraction from large collections of informal product descriptions, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, 2013, pp. 290–300.

[18] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, A. Egyed, An assessment of search-based techniques for reverseengineering feature models, Journal of Systems and SoftwareAccepted: to appear.

[19] L. Linsbauer, R. Lopez-Herrejon, A. Egyed, Feature model synthesis with genetic programming, in: C. Le Goues, S. Yoo (Eds.), Search-Based Software Engineering, Vol. 8636 of Lecture Notes in Computer Science, 2014, pp. 153–167.

[20] M. Harman, Y. Jia, J. Krinke, B. Langdon, J. Petke, Y. Zhang, Search based software engineering for software product line engineering: a survey and directions for future work (keynote paper), in: 18^{th} International Software Product Line Conference (SPLC 14), Florence, Italy, 2014, pp. 5–18.

[21] N. Andersen, K. Czarnecki, S. She, A. Wąsowski, Efficient synthesis of feature models, in: Proceedings of the 16th International Software Product Line Conference - Volume 1, SPLC '12, 2012, pp. 106–115.

[22] D. Benavides, S. Segura, P. Trinidad, A. Ruiz-cortés, FAMA, a framework for automated analyses of feature models, http://http://www.isa.us.es/fama/ (2014).

[23] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, A. Ruiz-Cortés, BeTTy:BEnchmarking and TestTing on the analYses of feature models, http://http://www.isa.us.es/betty/ (2014).

[24] M. Mendonca, M. Branco, D. Cowan, S.P.L.O.T.: Software product lines online tool, http://www.splot-research.org/, Generative Software Development Lab. Computer Systems Group, University of Waterloo (2014).

[25] C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, Cambridge, UK, 2008.
URL `http://nlp.stanford.edu/IR-book/information-retrieval-book.html`