Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An assessment of search-based techniques for reverse engineering feature models

Roberto E. Lopez-Herrejon[a,*], Lukas Linsbauer[a], José A. Galindo[b], José A. Parejo[b], David Benavides[b], Sergio Segura[b], Alexander Egyed[a]

[a] Institute for Software Systems Engineering, Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
[b] Department of Computer Languages and Systems, University of Seville, Av Reina Mercedes S/N, 41012 Seville, Spain

ARTICLE INFO

Article history:
Received 28 November 2013
Revised 10 July 2014
Accepted 14 October 2014
Available online 28 October 2014

Keywords:
Feature model
Reverse engineering
Search Based Software Engineering

ABSTRACT

Successful software evolves from a single system by adding and changing functionality to keep up with users' demands and to cater to their similar and different requirements. Nowadays it is a common practice to offer a system in many variants such as community, professional, or academic editions. Each variant provides different functionality described in terms of features. Software Product Line Engineering (SPLE) is an effective software development paradigm for this scenario. At the core of SPLE is variability modelling whose goal is to represent the combinations of features that distinguish the system variants using feature models, the de facto standard for such task. As SPLE practices are becoming more pervasive, reverse engineering feature models from the feature descriptions of each individual variant has become an active research subject. In this paper we evaluated, for this reverse engineering task, three standard search based techniques (evolutionary algorithms, hill climbing, and random search) with two objective functions on 74 SPLs. We compared their performance using precision and recall, and found a clear trade-off between these two metrics which we further reified into a third objective function based on F_β, an information retrieval measure, that showed a clear performance improvement. We believe that this work sheds light on the great potential of search-based techniques for SPLE tasks.

1. Introduction

Successful software evolves not only to adapt to emerging development technologies but also to meet the clients' and users' functionality demands. For instance, it is not uncommon to find academic, professional, or community variants (a.k.a editions) of commercial and open source applications where each variant provides different *features* increments in programme functionality (Zave).

The most common scenario in practice starts with a first system variant from which a new independent development branch is forked when a new variant with different feature combinations is required. This process is repeated as many times as new variants, also with different feature combinations, are requested. Unfortunately, this approach does not scale well as the number of features and their combinations increases even slightly (Krueger, 2001). *Software Product Line Engineering* (*SPLE*) is a software development paradigm devised to cope with the problems entailed by this scenario. SPLE advocates a disciplined yet flexible approach to maximize reuse and customization in all the software artefacts used throughout the entire development cycle (Krueger, 2001; Czarnecki and Eisenecker, 2000; Pohl et al., 2005; van d. Linden et al., 2007). The driving goal of SPLE is to create *software product lines (SPLs)* that realize the different software system variants in an effective and efficient manner.

However, developing SPLs from existing and individually developed system variants is not an easy endeavour. A crucial requirement is capturing all the feature combinations present in SPLs and represent them with *feature models* (*FMs*) (Czarnecki and Eisenecker, 2000; Kang et al., 1990), a de facto standard for modelling *variability* – the capacity of software artefacts to change (Svahnberg et al., 2005). Previous research has addressed this reverse engineering challenge from different perspectives with different approaches such as configuration scripts (She et al., 2011), propositional logic expressions (Czarnecki and Wasowski, 2007),

[*] Corresponding author. Tel.: +43 732 2468 4380.
E-mail addresses: roberto.lopez@jku.at (R.E. Lopez-Herrejon), lukas.linsbauer@jku.at (L. Linsbauer), jagalindo@us.es (J.A. Galindo), japarejo@us.es (J.A. Parejo), benavides@us.es (D. Benavides), sergiosegura@us.es (S. Segura), alexander.egyed@jku.at (A. Egyed).

http://dx.doi.org/10.1016/j.jss.2014.10.037
0164-1212/© 2014 Elsevier Inc. All rights reserved.

natural language (Weston et al., 2009), and ad hoc algorithms (Haslinger et al., 2011, 2013; Acher et al., 2012).

Our previous exploratory study analysed evolutionary algorithms for this reverse engineering task (Lopez-Herrejon et al., 2012). In this paper, we extend that work by:

- Including 15 new case studies of different domains.
- Employing two more search techniques, steepest ascent hill climbing and random search (Luke, 2009).
- Considering two new objective functions.
- Defining objective functions in terms of standard feature model operations.
- Extending the state representation with additional feature ordering information.
- Adding new mutation and crossover operators.
- Comparing and contrasting the objective functions using standard information retrieval metrics, recall and precision (Manning et al., 2008).
- Performing a detailed statistical analysis along the lines suggested by Arcuri and Briand (2014).

Our evaluation revealed a clear trade-off between recall and precision in our two objective functions. We further analysed this trade-off and reified it into an objective function that showed a clear improvement. We believe that this work is a stepping stone towards leveraging the wealth of Search-Based Software Engineering techniques for this and other SPLE challenges.

The structure of the paper is as follows. Section 2 provides the basic background on feature models and presents our running example. Section 3 describes the representation used to encode feature models. Section 4 presents the three search-based techniques under assessment and how they were adapted to our problem. Section 5 describes the objective functions analysed in our study and the definitions of recall and precision metrics for our reverse engineering task. Section 6 presents a short overview of our implementation. Section 7 describes how the evaluation was carried out to compare and contrast the three algorithms with our two objective functions, analyses the results obtained highlighting the trade-off we found between precision and recall, and defines and evaluates the third objective function that reifies this trade-off. Section 8 describes the threats to validity identified in our work and how they were addressed. Section 9 summarizes the related work closest to ours. Section 10 highlights some open issues for future work, and Section 11 summarizes our conclusions.

2. Running example and feature models

As a running example let us consider a hypothetical set of variants of a software system for controlling Video On Demand (VOD) services for home and personal entertainment. In this example, there are 18 different variants depicted in Table 1. For each variant the set of features that are selected are depicted with tick marks √. For sake of brevity, we employ abbreviations in the feature header labels. All the systems have a common functionality (e.g. turning on/off) and can play shows. These two features are respectively denoted as VOD and Play in the table. Do notice that both are selected in all the system variants. Similarly all systems have displaying capability, denoted by feature Display, and have an operating system (feature OS) with its kernel (feature Ker). Some systems have: recording capability (feature Rec), can be used either in a TV set (feature TV) or in a mobile device (feature Mob) which can be standard phone sets (feature Std) or smart phone sets (feature Sm), advanced operating systems capability (feature Adv), areal antenna (feature Aer), cable TV capability (feature Cab), and pay-per-view (feature PPV).

Our reverse engineering process starts from the set of variants and their provided features, as captured in a table like Table 1, and has as goal obtaining a feature model that represents such feature combinations. Recall that feature models are the de facto standard to model relations among the features and thus the common and variable features of an SPL (Kang et al., 1990). In feature models, features are depicted as labelled boxes and are connected with lines to other features with which they relate, collectively forming a tree-like structure. A feature can be classified as:

- *Mandatory feature.* A mandatory feature is selected in a system whenever its parent feature is also selected. It is depicted with a filled circle at the child end of the feature relation. For example, Fig. 1a illustrates mandatory feature B.
- *Optional feature.* An optional feature may or may not be part of a programme whenever its parent feature is part. It is depicted with an empty circle at the child end of the feature relation. Fig. 1b is an example of an optional feature B.

Features can also be grouped into:

- *Alternative groups.* If the parent feature of the group is selected, *exactly* one feature from the group must be selected. Alternative groups are depicted with lines connecting the parent feature with the group features and an empty arc crossing the lines. For example, Fig. 1c illustrates that if feature P is selected, then one of the group features C1, C2 or C3 must be selected.
- *Or groups.* If the parent feature of the group is selected, then *one or more* features from the group can be selected. Or groups are depicted with lines connecting the parent feature and the group features plus a filled arc crossing the lines. Fig. 1d shows that if feature P is selected, one of more of features C1, C2 or C3 must be selected. For instance, the combination of C1 with C2, or the combination that has all three group features C1, C2 and C3.

Besides the parent–child relations, features can also relate across different branches of the feature model with *cross-tree constraints* (*CTCs*) (Benavides et al., 2010). The typical examples of this kind of relations are: (i) *requires* relation whereby if a feature A is selected a feature B must also be selected, and (ii) *excludes* relation whereby if a feature A is selected then feature B *must not* be selected and vice versa. In a feature model, these latter relations are depicted with doted single-arrow lines and dotted double-arrow lines respectively. Fig. 1e illustrates these kinds of CTCs. Additionally, more general cross-tree constraints can be expressed using propositional logic (Benavides et al., 2010).

The reverse engineering process of our work is summarized in Fig. 2, which shows as input the system variants with their selected features. By using search-based techniques, our goal is to obtain a feature model that captures the feature combinations of the system variants. It should be pointed out that feature models are non-canonical representations, meaning that in general a set of feature combinations could be represented by more than one different feature model. Thus trade-offs between different solutions can be found. In addition, the commonly large number of features and number of variants makes it a problem suitable for search-based techniques.

We should remark that reverse engineering a SPL from a set of system variants is indeed an iterative process whereby all the involved stakeholders go through multiple iterations where the feature models, SPL architecture and supporting platform are successively refined. The goal of our work is to provide a first working feature model which can subsequently be refined through this iterative process mentioned.

As an example target of our reverse engineering process let us consider a feature model for our running example shown in Fig. 3.

Table 1
System variants of VOD software product line.

Variant	VOD	Play	Rec	Disp	OS	TV	Mob	Sm	Std	Ker	Adv	Aer	Cab	PPV
V1	√	√	√	√	√	√				√			√	√
V2	√	√		√	√	√				√			√	√
V3	√	√	√	√	√	√				√		√		
V4	√	√		√	√	√				√		√		
V5	√	√	√	√	√		√		√	√				√
V6	√	√	√	√	√		√		√	√				
V7	√	√		√	√		√		√	√				
V8	√	√		√	√		√		√	√				√
V9	√	√	√	√	√	√				√	√		√	√
V10	√	√		√	√	√				√	√		√	√
V11	√	√	√	√	√	√				√	√	√		
V12	√	√		√	√	√				√	√	√		
V13	√	√	√	√	√		√		√	√	√			√
V14	√	√	√	√	√		√		√	√	√			
V15	√	√		√	√		√		√	√	√			√
V16	√	√		√	√		√		√	√	√			
V17	√	√	√	√	√		√	√		√	√			√
V18	√	√		√	√		√	√		√	√			√

Feature abbreviations: video on demand (VOD), play (Play), record (Rec), display (Disp), operating system (OS), television (TV), mobile (Mob), smart phone (Sm), standard phone (Std), kernel (Ker), advance OS (Adv), aereal TV (Aer), cable TV (Cab), pay-per-view (PPV).

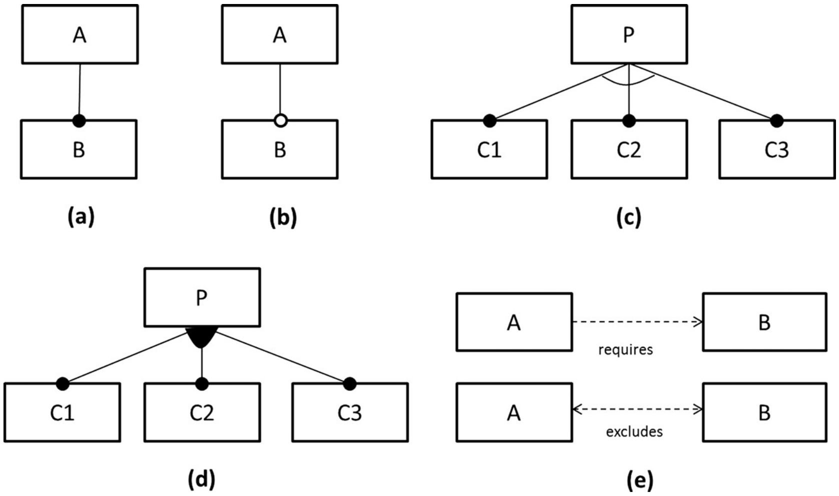
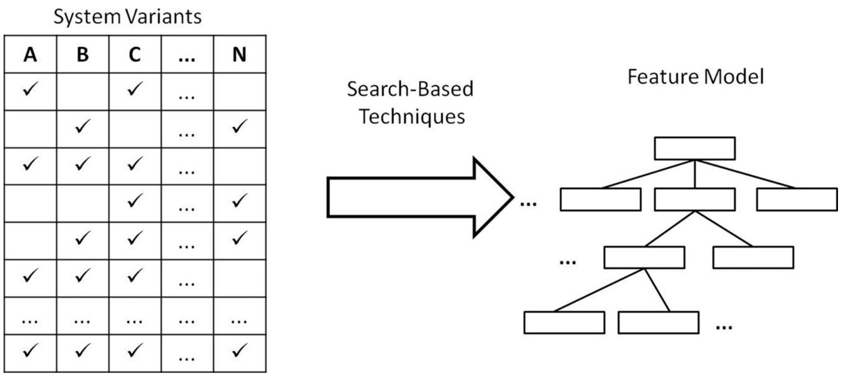


Fig. 1. Feature models graphical notation.

We can pick as the root feature VOD because it is always present in all systems. Features Play, Display and OS are mandatory, whereas Record and PPV are optional. In addition, feature OS has mandatory feature Kernel and optional feature Advanced. This feature model contains three alternative groups: (i) for the kinds of displays, features Display with TV and Mobile, (ii) for types of TV input, feature TV with Aerial or Cable, and (iii) for types of mobiles, features Mobile with Standard and Smart. Focus now on the CTCs. Notice the requires relations between features Smart and Advanced, meaning that if a product contains feature Smart it must also contain feature Advanced. The same is the case between Smart and PPV, and between Cable and PPV. Finally, features Aerial and PPV


System Variants

A	B	C	...	N
✓		✓	...	
	✓		...	✓
✓	✓	✓	...	
		✓	...	✓
	✓	✓	...	✓
✓	✓	✓	...	
...	...	...	...	...
✓	✓	✓	...	✓

Fig. 2. Reverse engineering feature models overview.

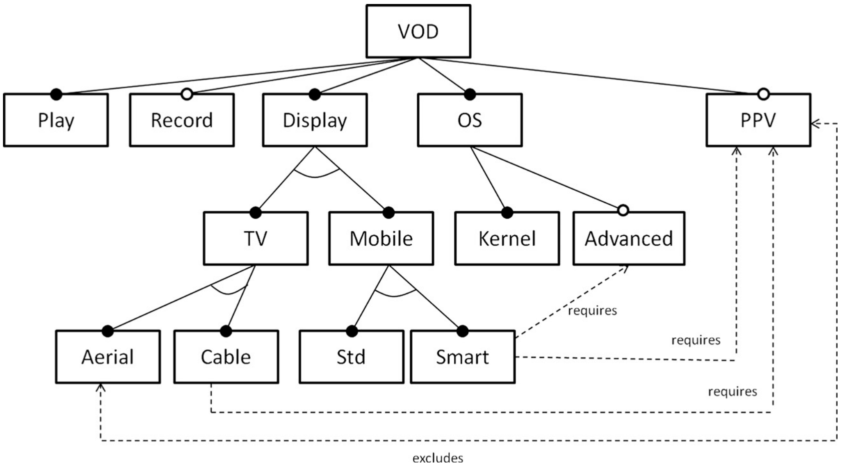
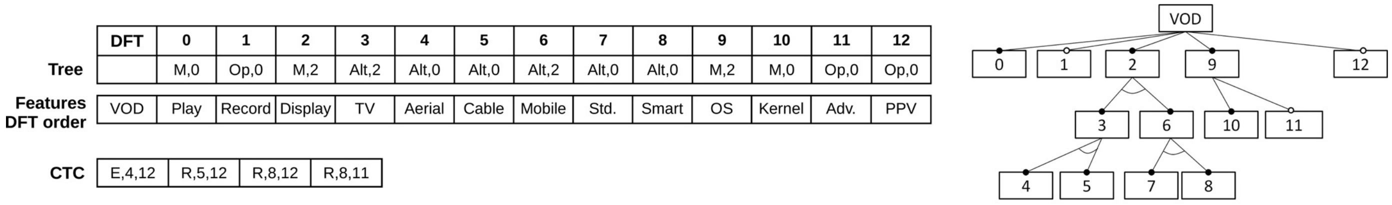


Fig. 3. Video on demand SPL feature model.



	DFT	0	1	2	3	4	5	6	7	8	9	10	11	12
Tree		M,0	Op,0	M,2	Alt,2	Alt,0	Alt,0	Alt,2	Alt,0	Alt,0	M,2	M,0	Op,0	Op,0
Features DFT order	VOD	Play	Record	Display	TV	Aerial	Cable	Mobile	Std.	Smart	OS	Kernel	Adv.	PPV

CTC	E,4,12	R,5,12	R,8,12	R,8,11

(a) Arrays (b) Tree Traversal

Fig. 4. ETHOM-based search state representation of feature models.

are in an `excludes` relation meaning that both cannot be present in the same product.

Next we provide a more formal definition of feature sets and their relation with system variants.[1]

Definition 1. A *feature set* is a 2-tuple $[\mathtt{sel},\overline{\mathtt{sel}}]$ where `sel` and $\overline{\mathtt{sel}}$ are respectively the set of selected and not-selected features of a system variant. Let `FL` be the list of features of a feature model, such that `sel`, $\overline{\mathtt{sel}} \subseteq \mathtt{FL}$, $\mathtt{sel} \cap \overline{\mathtt{sel}} = \emptyset$, and $\mathtt{sel} \cup \overline{\mathtt{sel}} = \mathtt{FL}$.

Definition 2. A feature set is *valid* if the selected and not-selected features adhere to all the constraints imposed by the feature model.

For example, the feature set `fs=[{VOD, Play, Display, TV, Aerial, OS, Kernel}, {Record, Cable, Mobile, Std, Smart, Advanced, PPV}]` is valid. In fact, it corresponds to variant V4 in Table 1. As another example, a feature set with features `TV` and `Mobile` is not valid because it violates the constraint of the or relation which establishes that these two features cannot appear selected together in the same feature set.

3. Search space representation

The states of our search space are the different feature models that can be formed with all the features in the system variants from which a feature model is going to be reverse engineered. We base our state representation on ETHOM, which stands for*Evolutionary algoriTHm for Optimized feature Models* (Segura et al., 2014). In this work a feature model is represented as two arrays, one for the tree structure of the feature model and one for the CTCs. A limitation of this representation is that the order of features is fixed. Every index in the tree structure array is assigned a fixed feature at the very beginning. To remove this limitation we extend the representation by an additional array that represents the depth-first traversal order of the features. Let us now describe the three arrays of the extended ETHOM representation.

The array that represents the structure of a feature model tree consists of tuples of the form ⟨`PR`, `CN`⟩ where:

- `PR` denotes the type of relationship a feature has with its parent. It can be `M` for mandatory, `Op` for optional, `Alt` for alternative, and `Or` for or relation.
- `CN` denotes the number of children of the feature.

The order of these tuples in this array is determined by a depth-first traversal (DFT) of the feature model starting from the root. Fig. 4 shows the DFT traversal of the feature model of our running example shown in Fig. 3. Notice that the root is not encoded. As a first example consider the tuple at entry with DFT value 6. It encodes feature `Mobile` which is an alternative (`PR` value is `Alt`) feature of its parent (feature `Display`), and has two children (`CN` value is 2), namely `Std` and `Smart`. As another example, consider the tuple at entry with DFT value 11. This tuple encodes feature `Advanced`, with an optional relation with its parent feature `OS` (`PR` value is `Op`), and with no children (`CN` value is 0). The complete array for the structure of the feature model is shown in Fig. 4a.

The second array determines which feature is assigned to which DFT index and also which feature is the root. In Fig. 4a the array for the DFT order of the features is shown beneath the array for the tree structure. The feature order array contains one additional element because it also encodes the root feature. Feature *VOD* is the root feature. DFT index 0 is assigned feature *Play* and so on.

The array of the cross-tree constraints are tuples of the form ⟨`TC`, `O`, `D`⟩ where:

[1] Adapted from Benavides et al. (2010) where feature sets are referred to as configurations.

- TC encodes the type of cross-tree constraint. An R value denotes a requires constraint whereas an E value denotes an excludes constraint.
- O encodes the origin feature of the cross-tree constraint represented with the corresponding DFT value.
- D encodes the destination feature of the cross-tree constraint represented with the corresponding DFT value.

For example, the tuple ⟨E, 4, 12⟩ encodes the excludes cross-tree constraint between features Aerial (DFT value 4) and PPV (DFT value 12). As another example, the tuple ⟨R, 8, 11⟩ encode a requires cross-tree constraint between features Smart (DFT value 8) and Advanced (DFT value 11). The complete array for the CTCs of the feature model is shown in Fig. 4a.

4. Search based techniques

In this section we sketch the search techniques we use in our study, their operators and their parameters setting. In Section 6 we describe how they were actually realized.

4.1. Evolutionary algorithm (EA)

Our previous work explored a evolutionary algorithm for reverse engineering feature models which we summarize in this section (Lopez-Herrejon et al., 2012). It follows the standard evolutionary algorithm flow that starts with a randomly generated initial population of individuals, in our case feature models, which are then evaluated. The next population is subsequently computed by *selecting* individuals from the current population and randomly performing *crossover* and *mutation* operations on them. This process is repeated until an ideal individual is found or until the maximum number of generations is reached. Next we explain how the initial population was created, and how crossover and mutation work in our context.

4.1.1. Initial population and selection

There are different alternatives in the literature to randomly generate feature models (Thüm et al., 2009; Segura et al., 2012). We relied on BeTTy, a framework that implements ETHOM and can generate random feature models with the following configuration parameters: (i) population size, (ii) number of features, (iii) percentage of cross-tree constraints (relative to the number of features), (iv) maximum branching factor (defined as the maximum number of subfeatures of a feature, considering all the types of relationships), (v) percentage of mandatory relations, (vi) percentage of optional relations, (vii) percentage of Alternative relations, (viii) percentage of Or relations. We chose standard values to set the parameters for the generation of the initial population of feature models, see Table 2. Similarly, we used standard values for the parameters of the evolutionary algorithm as shown in Table 3.

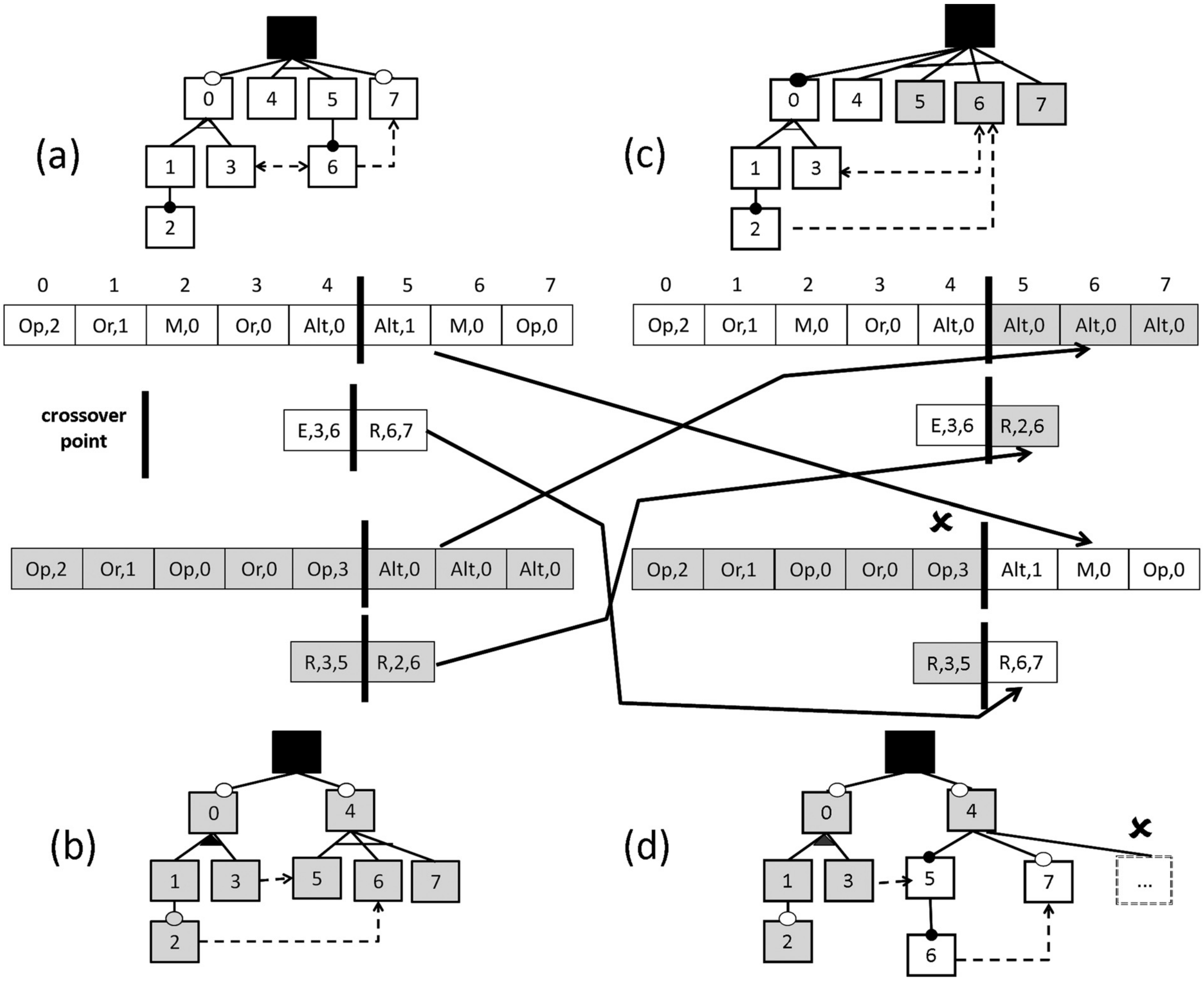

(a)
0
4
5
7
1
3
6
2
(c)
0
4
5
6
7
1
3
2
0 1 2 3 4 5 6 7
Op,2 Or,1 M,0 Or,0 Alt,0 Alt,1 M,0 Op,0
Op,2 Or,1 M,0 Or,0 Alt,0 Alt,0 Alt,0 Alt,0
crossover point
E,3,6 R,6,7
E,3,6 R,2,6
Op,2 Or,1 Op,0 Or,0 Op,3 Alt,0 Alt,0 Alt,0
Op,2 Or,1 Op,0 Or,0 Op,3 Alt,1 M,0 Op,0
R,3,5 R,2,6
R,3,5 R,6,7
(b)
0
4
1
3
5
6
7
2
(d)
0
4
1
3
5
7
...
2
6

Fig. 5. Example of one-point crossover in ETHOM-based representation (Segura et al., 2014).

Table 2
Configuration parameters for initial population generation.

Parameter	Value selected
Population size	100
Number of features	Same as input products
Percentage of CTCs	0
Maximum branching factor	10
Percentage of mandatory relations	Random[a]
Percentage of optional relations	Random[a]
Percentage of `Alt` relations	Random[a]
Percentage of `Or` relations	Random[a]

[a] Random such that they sum up to 1.

Table 3
Summary of evolutionary algorithm configuration parameters.

Parameter	Value selected
Selection strategy	Roulette-wheel
Crossover strategy	One-point
Crossover probability	0.7
Per gene mutation probability	0.01
Per chromosome mutation probability	0.1
Population size	100
Infeasible individuals	Repair
Maximum generations	25

There are multiple alternatives to implement selection in the evolutionary algorithms literature (Eiben and Smith, 2003). We chose roulette wheel as our selection strategy, again, a standard setting in evolutionary algorithm research.

4.1.2. Operators

The two characteristic operators of the ETHOM-based representation are crossover and mutation. We describe and illustrate how they work next.

4.1.2.1. Crossover. There are multiple alternatives to implement crossover (Eiben and Smith, 2003). For the crossover of the tree structure array and the CTC array we adapted the one-point crossover whose application is illustrated in Fig. 5. The process starts by selecting two parent individuals, in our case feature models encoded as described in Section 3, like those shown in Fig. 5a and b. The crossover point is depicted with a bar symbol (|). Like standard crossover, the two new offsprings are generated by swapping the tuples starting from the crossover point on both the structural and the CTCs arrays. For example, the first offspring shown in Fig. 5c, has in its structural array the tuples DFT-indexed 0–4 from the first parent (Fig. 5a) and those DFT-indexed 5–7 from the second parent (Fig. 5b). Similarly, the CTCs array of this new offspring has its first tuple from the first parent and the second tuple from its second parent.

The crossover of the feature order array is not as simple, because additional constraints apply here. Every feature must appear exactly once in every feature model individual. However, a

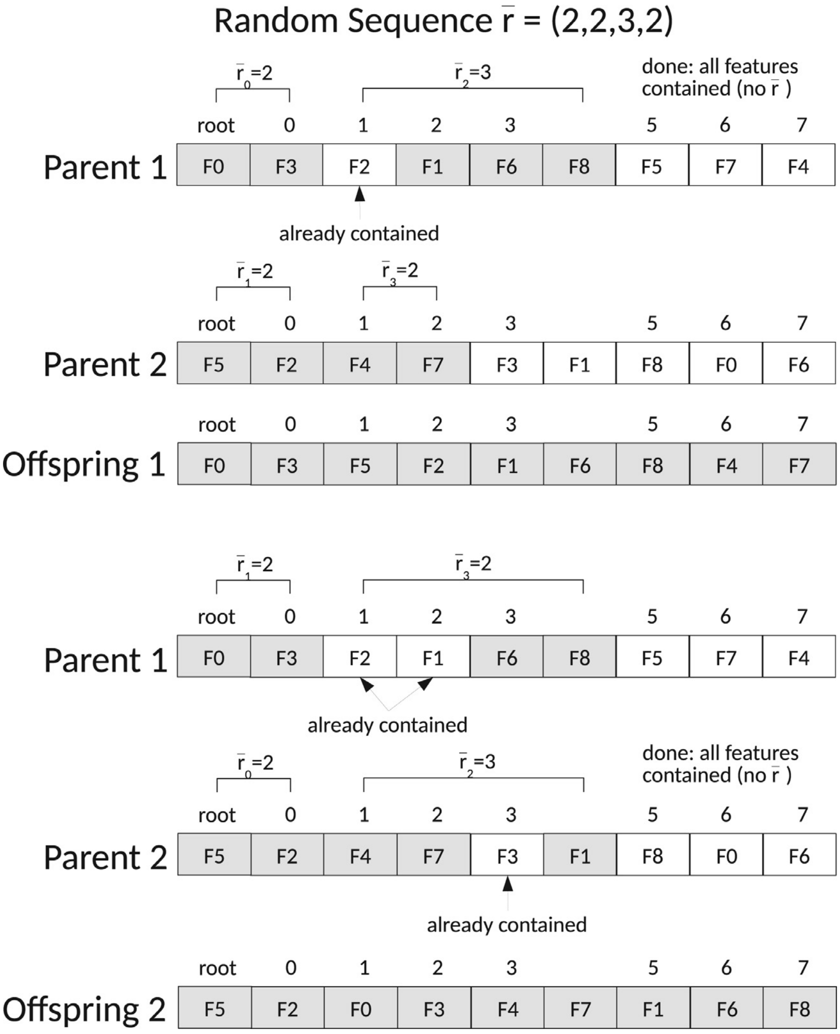


Fig. 6. Feature order crossover.

simple one-point crossover will violate this constraint in that features will appear twice or not at all. Therefore we devised another crossover operator specifically designed for this purpose. It first generates a sequence of random numbers $\bar{r}$ that sum up to the number of features in the feature model. Then it takes the first $\bar{r}_0$ features from the first parent and puts them into the offspring, then $\bar{r}_1$ from the second parent, $\bar{r}_2$ from the first parent again and so on. Features that are already contained in the offspring are skipped and not counted. For the second offspring the same thing is done using the same random numbers sequence $\bar{r}$ but starting with the second parent. An example is shown in Fig. 6 for the random number sequence $\bar{r} = \langle 2, 2, 3, 2 \rangle$. The first $\bar{r}_0 = 2$ features F_0 and F_3 are taken from the first parent and put into the offspring. Then $\bar{r}_1 = 2$ features, F_5 and F_2, are taken from the second parent. For the next $\bar{r}_2 = 3$ features from the first parent the feature F_2 is skipped because it is already contained in the offspring and features F_1, F_6 and F_8 are put into the offspring. Lastly the remaining $\bar{r}_3 = 2$ features, F_4 and F_7, from the second parent are copied over and the crossover for the first offspring is done. The same thing is repeated for the second offspring only this time starting with the second parent.

Performing the crossover operation can yield infeasible individuals, i.e. not well-formed feature models. Consider for instance the second offspring produced shown in Fig. 5d. Notice that the tuple with DFT-index 4 indicates that this feature must have three children features. However, the next tuple (DFT-index 5) indicates that the corresponding feature must also have a child feature. But this is not possible because there are not enough features in the part crossed over to satisfy both of these requirements. We employ ETHOM mechanisms for the identification and repair of feature model errors. A short overview is presented in Appendix A, for further details please refer to Segura et al. (2014).

4.1.2.2. Mutation. We apply mutations either on a per gene (i.e. array entry) basis where every gene in a chromosome has a chance of being mutated or on a per chromosome basis where a chromosome has a chance of being mutated. Next we describe the mutation operators that we employed.

For the tree-structure array:

- *Operator* 1. Changes randomly a relation between two features from one kind to any other kind. For example, from mandatory (`M`) to optional (`Op`) or from `Op` to `Alternative` (`Alt`). This mutation is applied per gene.
- *Operator* 2. Changes the number of children `CN`, to a number selected from 0 to a maximum branching factor parameter. This mutation is applied per gene.

For the features DFT order array:

- *Operator* 3. Two random features of a chromosome are swapped in the features DFT order array.

For the CTC array:

- *Operator* 4. Changes the type of cross-tree constraint, from excludes to requires and vice versa. This mutation is applied per gene.
- *Operator* 5. Changes either the origin or destination feature (with equal probability) of a cross-tree constraint. It is checked that the resulting CTC does not have the same origin and destination values. This mutation is applied per gene.
- *Operator* 6. Removes a cross-tree constraint. This mutation is applied per gene.
- *Operator* 7. Adds a random cross-tree constraint to a chromosome.

Like crossover, the mutation operators can produce feature models that are not semantically correct. As before, we relied on ETHOM mechanisms to identify and repair feature models with errors (Segura et al., 2014). Table 3 summarizes the configuration parameters for the genetic algorithm and the values that were used.

4.2. Steepest ascent hill climbing (HC)

We employed a standard steepest ascent hill climbing search as sketched in Algorithm 1 (Luke, 2009). It starts with a randomly generated initial state (Line 1). From this state, a neighbourhood of `SampleSize` states is created and searched (Lines 6–12). If a neighbour state has a better fitness value than the current state, the search moves to that state (Lines 13–14). This process is repeated until the number of iterations (`MaxIter`) is exceeded or an ideal solution (`BestFitness`) is reached (Line 4).

Algorithm 1 Steepest Ascent Hill Climbing

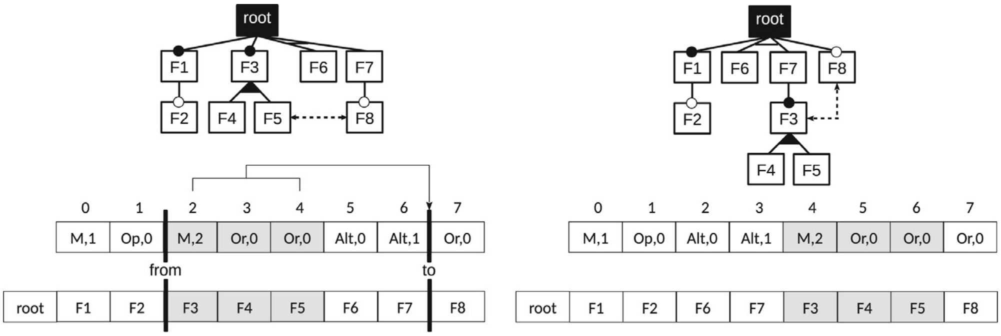
```
 1: $X \leftarrow$ random initial state
 2: $I \leftarrow 0$
 3: $Best \leftarrow X$
 4: while $(I < MaxIter) \wedge (evaluate(Best) \neq BestFitness)$ do
 5:     $S \leftarrow 0$
 6:     while $S < SampleSize$ do
 7:         $X' \leftarrow move(Best)$
 8:         if $evaluate(X')$ better than $evaluate(X)$ then
 9:             $X \leftarrow X'$
10:         end if
11:         $S \leftarrow S + 1$
12:     end while
13:     if $evaluate(X)$ better than $evaluate(Best)$ then
14:         $Best \leftarrow X$
15:     end if
16:     $I \leftarrow I + 1$
17: end while
18: return $Best$
```

Note here the auxiliary function *move* in Line 7. This function receives as input a state (i.e. an encoded feature model) and applies the mutation operators, described in Section 4.1.2, in combination with a new operator *shuffle* to generate a valid new state. This new operator was added to provide more diversity in the generated neighbour states. It works by randomly generating: (i) number of tuples to shuffle, (ii) an index number in the array where the tuples are selected from, and (iii) an index number in the array where the tuples are moved to. Fig. 7 illustrates this operator. First it indicates in Fig. 7a the three required random numbers. In this example, it shuffles three tuples, from array index 2 to array index 7 in the tree structure array along with the corresponding features in the feature order array. The resulting feature model and its encoding are shown in Fig. 7b.

Notice that the shuffle operator is not applicable to the CTC array of the feature model encoding because there the tuple order does not matter. The parameters of this search technique and their corresponding values used are summarized in Table 4.

4.3. Random search (RS)

We follow a standard random search as sketched in Algorithm 2. It starts with a random initial state (Line 1). A new random state is generated, using the *move* auxiliary function as explained before (Line 4). The search moves to a new state if it has a fitness value better than the current `Best` state (Lines 5–6). This process is repeated


root
F1
F2
F3
F4
F5
F6
F7
F8
0 1 2 3 4 5 6 7
M,1 Op,0 M,2 Or,0 Or,0 Alt,0 Alt,1 Or,0
from
to
root F1 F2 F3 F4 F5 F6 F7 F8
M,1 Op,0 Alt,0 Alt,1 M,2 Or,0 Or,0 Or,0
root F1 F2 F6 F7 F3 F4 F5 F8

(a) Before applying shuffle

(b) After shuffle application

Fig. 7. Shuffle operator example.

Table 4
Steepest ascent hill climbing configuration parameters.

Parameter	Value selected
Shuffle probability	0.3
Per tuple mutation probability	0.1
Samples size	100
Infeasible individuals	Repair
Maximum iterations	25

until the number of iterations (`MaxIter`) is exceeded or an ideal solution (`BestFitness`) is reached (Line 3). For a fair comparison, the number of iterations is set to generate the same number of states (calls to function *move*) like the other two search techniques of our study. Thus this value was set to 100 × 25 = 2500 iterations.

Algorithm 2 Random Search

```
1:  $Best \leftarrow X$ random initial state
2:  $I \leftarrow 0$
3:  while $(I<MaxIter) \wedge (evaluate(Best) \neq BestFitness)$ do
4:      $X \leftarrow$ random candidate solution
5:      if $evaluate(X)$ better than $evaluate(Best)$ then
6:          $Best \leftarrow X$
7:      end if
8:      $I \leftarrow I + 1$
9:  end while
10: return $Best$
```

5. Objective functions

In this section we define the two objective functions used in our study. These functions were defined in terms of feature model analysis operations as shown next. Subsequently, we describe the two standard information retrieval metrics used to compare and contrast the results obtained.

5.1. Auxiliary functions

The coming auxiliary functions were implemented based on the feature model analysis operations supported by FaMa (Benavides et al., 2007), whose definitions are described by Benavides et al. (2010).

Let $\mathcal{FM}$ denote the universe of feature models, and $\mathcal{SFS}$ the universe of sets of feature sets.

- *#features* : $\mathcal{FM} \rightarrow \mathbb{N}$, returns the number of features contained in a feature model.
- *#featureSets* : $\mathcal{FM} \rightarrow \mathbb{N}$, returns the number of feature sets (a.k.a products) denoted by a feature model.
- *#containedFeatureSets* : $\mathcal{SFS} \times \mathcal{FM} \rightarrow \mathbb{N}$, returns the number of feature sets received as first argument `sfs` that are valid according to a feature model `fm`.
- *surplus* : $\mathcal{SFS} \times \mathcal{FM} \rightarrow \mathbb{N}$, returns the number of feature sets denoted by feature model `fm` that are not contained in the required feature sets `sfs`:

$$\begin{aligned} surplus(sfs, fm) = \#featureSets(fm) \\ -\#containedFeatureSets(sfs, fm) \end{aligned} \quad (1)$$

It holds that $surplus(sfs, fm)\ \ 0$.
- *deficit* : $\mathcal{SFS} \times \mathcal{FM} \rightarrow \mathbb{N}$, returns the number of feature sets in `sfs` that are not contained in the feature sets denoted by feature model `fm`:

$$deficit(sfs, fm) = |sfs| - \#containedFeatureSets(sfs, fm) \quad (2)$$

It holds that $0 \leq\ deficit(sfs, fm) \leq |sfs|$.

5.2. Objective functions definitions

We devised two objective functions to capture the different reverse engineering concerns for feature models. Next we define each of these functions and explain their underlying motivation.

Relaxed objective function (Relaxed). This function expresses the concern of capturing primarily the feature sets of the system variants (function # *containedFeatureSets*), any other feature sets that could be denoted by the reverse-engineered feature model are thus irrelevant. Hence, this objective function should be maximized, with the maximum value being the number of feature sets to reverse engineer. Equation 3 shows the definition of this function.

$$RelaxedFF(sfs, fm) = \#containedFeatureSets(sfs, fm) \quad (3)$$

Minimal difference objective function (MinDiff). Based on our previous experience with RelaxedFF, we observed that this function indeed retrieved the desired feature sets in the majority of cases (Lopez-Herrejon et al., 2012). However, it also retrieved many more feature sets that were not relevant. The goal of MinDiff is precisely to express the concern of obtaining a closer-fit feature model from the feature sets of the system variants. MinDiff accounts for the number of feature sets missing (function `deficit`) and the number of additional feature sets (function `surplus`) in the reverse-engineered feature model. This objective function should be minimized, with the minimum value being zero which represents the perfect fit. Eq. 4 shows the definition of this function.

$$minDiffFF(sfs, fm) = deficit(sfs, fm) + surplus(sfs, fm) \quad (4)$$

Table 5
Feature models summary.

Feature model name	NF	NFS	Domain	Feature model name	NF	NFS	Domain
model_20110519_1142211980	10	1	Container	ZipMe	8	64	Data compression
model_20110301_216655728	10	2	Automotive	model_20110130_639381749	12	64	Insurance
model_20120328_523540818	10	8	Phone	model_20110203_1382675896	14	64	OCL tool
model_20120113_1950870026	10	9	e-Shop	PKJab	12	72	Messenger
model_20110601_1103941862	12	11	Text editor	model_20110401_868452735	14	72	Academic
model_20091015_449909368	10	12	Language	gpl	18	73	Graph algorithms
car_fm	9	1 3	Academic	REAL-FM-7	14	80	IDE tool
model_20120328_1373483522	13	14	Parking lot	model_20091129_1734444143	12	84	Tutoring system
model_20120110_1719396361	10	16	Phone	model_20111129_1932950448	13	96	NA
model_20120110_855603964	10	16	Academic	model_20110406_656545830	17	96	Audio player
model_20091219_494647199	11	16	Web portal	model_20110605_1465859222	17	112	Mobile media
model_20110915_1159959623	10	17	academic	model_20111020_455520177	11	120	Robotics
connector_fm	20	18	Academic	model_20110323_789959080	15	120	Software pattern
model_20111027_1380540076	10	20	NA	model_20110823_553386338	21	128	Academic
model_20120110_1754443954	10	20	Phone	model_20091206_1647557456	10	135	Game
REAL-FM-13	12	21	Telecommunications	argo-uml-spl	11	192	UML tool
model_20101117_1571856147	10	24	Economy	model_20110306_314567479	14	248	TV set
model_20120202_1596034358	10	28	Mobile	BDBFootprint	9	256	Database
model_20110622_260389190	12	28	Automotive	Apache	10	256	Web server
REAL-FM-8	17	28	Monitoring system	aircraft_fm	13	315	Academic
model_20100927_1382418986	10	31	Mobile	fame_dbms_fm	20	320	Database
Prevayler	6	32	Object persistence	fame_dbms_fm_v2	21	320	Database
model_20101104_260803083	11	32	NA	model_20110925_62365838	23	336	Academic
model_20110519_503436691	11	35	NA	model_20100607_746327867	20	448	Academic
model_20100822_281357717	15	36	Web caching	DesktopSearcher	22	462	File search
model_20120109_1808102333	18	36	Software process	model_20120201_899753062	27	560	Academic
model_20111220_1184087779	15	40	Automotive	model_20090801_1908323193	13	624	Software library
model_20101123_920943759	15	44	Sales management	model_20110216_608697455	14	640	NA
model_20101111_1790887308	10	48	Phone	model_20100712_329430908	20	640	Card products
model_20100325_298677687	11	48	Video app	model_20110310_1849309646	23	810	Server app
model_20101111_156018899	12	48	e-Shop	model_20110203_2097159983	18	896	OCL tool
REAL-FM-10	14	48	Virtual office	LLVM	12	1024	Compiler library
model_20110823_1386366267	15	50	Software process	Curl	14	1024	Data transfer
model_20101124_661702924	15	60	Academic	LinkedList	27	1344	Data structures
model_20120328_361613983	16	60	Web game	x264	17	2048	Video encoding
model_20100308_1032655961	15	61	Mobile phone	BDBMemory	19	3840	Database
model_20111025_1408959776	13	63	Music player	Wget	17	8192	File retrieval

NF: number of features, NFS: number of feature sets, NA: domain not available.

5.3. *Comparing objective functions*

In order to compare the two objective functions we relied on two basic information retrieval metrics (see Manning et al., 2008). To the best of our knowledge, our work is the first to use these metrics in the context of reverse engineering feature models. Next we describe how they were adapted to our context.

Precision. The fraction of the retrieved feature sets that are relevant to the search.

$$precision(sfs, fm) = \frac{\#containedFeatureSets(sfs, fm)}{\#featureSets(fm)} \qquad (5)$$

Recall. The fraction of the feature sets that are relevant to the search that are successfully retrieved.

$$recall(sfs, fm) = \frac{\#containedFeatureSets(sfs, fm)}{|sfs|} \qquad (6)$$

In the subsequent sections we describe how our reverse engineering framework was implemented and evaluated.

6. System architecture and implementation

We based the implementation of our work on ETHOM (Segura et al., 2014) and the implementation of our own previous work (Lopez-Herrejon et al., 2012). We also used *FaMa (FeAture Model Analyser)* (Benavides et al., 2007, 2013) to implement the objective functions and other reasoning operators. FaMa is a pioneering and leading Java-based tool suite for the analysis of feature models. It provides a large catalogue of efficient implementations of feature model operations which can be easily extended. Furthermore, it supports multiple reasoning engines such as SAT or Constraint Programming tools. Additionally we relied on BeTTy, a benchmark and testing framework for feature model analysis (Segura et al., 2012). BeTTy provides an implementation of various tools for evolutionary algorithms with feature models like the conversion of a feature model into a chromosome or the generation of random feature models. ETHOM, which can generate feature models, for instance, that are hard to analyse in execution time or memory footprint (Segura et al., 2012) is based on BeTTy. An overview of the system architecture is shown in Fig. 8.

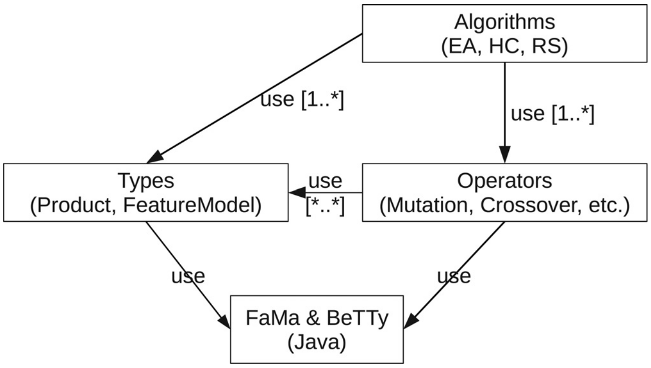


Fig. 8. Architecture overview.

Table 6
Average runtime per feature model in mins:secs.

	Relaxed	MinDiff
EA	02:26	01:56
HC	01:19	01:21
RS	01:16	02:28

Table 7
Average recall and precision values for relaxed and MinDiff.

	Recall			Precision		
Function	EA	HC	RS	EA	HC	RS
Relaxed	0.9649	0.8015	0.6975	0.1365	0.2251	0.1462
MinDiff	0.3307	0.2953	0.2189	0.6525	0.6637	0.4807

7. Evaluation

In this section we describe how our evaluation was carried out, the results obtained for each objective function, the statistical analysis performed and the conclusions that can be drawn.

7.1. Experimental setup

In practice, reverse engineering of SPL occurs usually with a low number of existing variants. For example, Hetrick et al. describe an industrial case study that was reverse engineered from 23 products (Hetrick et al., 2006). We selected for our work case studies with about a thousand feature sets.[2] This selection allows us on one hand to evaluate typical sizes of reverse engineered SPLs, but also allows us to evaluate scalability issues in larger samples.

Our previous work (see Lopez-Herrejon et al., 2012) considered 59 feature models selected from the SPLOT repository (see (Generative Software Development, 2013)). To this set we added 15 new feature models from actual SPLs that are publicly available.[3] This extended corpus now ranges from 6 to 27 number of features, and from 1 to 8192 number of feature sets. Table 5 presents a summary of the feature models selected.

For each feature model, we computed the feature sets that it denotes and saved them in a table that was used as input to our search based techniques, as illustrated in Fig. 2. We executed 30 independent runs for each feature model input set for each of the three algorithms and for each objective function. The total number of independent runs is thus: 74 (feature models) × 3 (algorithms) × 2 (objective functions) × 30 runs = 13,320.

We ran our experiments on an array of different machines with 4–16 cores with clock speeds between 2 and 4 GHz with 4–16 GBs of memory. A summary of the average runtimes for a single reverse engineering run of a single feature model is shown in Table 6.

7.2. Analysis

In this section we analyse the results obtained for each objective function and search algorithm whose average values of recall and precision are shown in Table 7, while the average values obtained for each feature model are shown in Fig. 9.

7.2.1. Results of Relaxed objective function

For the Relaxed objective function all three techniques ranked well according to the recall performance indicator with EA (0.96) in the first position followed by HC (0.80) and RS (0.69) respectively. This result means that the generated feature models included most of the input feature sets. In fact, EA generated feature models including the complete target set of products in 46 out of 74 problem instances, i.e. those with a recall value equal to 1. We used Pearson's coefficient to see if there was any correlation between the number of feature sets and the recall values. For EA (−0.25) it indicated a weak inverse relation (i.e. lower recall as the number of feature sets increases), whereas for HC (−0.40) and RS (−0.46), it indicated that recall values tend to decrease as the number of feature sets increases but not with a strong correlation.

Regarding precision, HC (0.22) returned the best average value followed by RS (0.14) and EA (0.13). In contrast to the recall results, the precision values are considerably low which means that most generated models contained a large amount of products not included in the input feature sets. Again we applied Pearson's correlation coefficient and did not find a strong correlation (all values <0.06). These results corroborate the fact that the Relaxed objective function favours recall while practically disregarding precision. This is clearly observed in Fig. 9a, c and e which show the values of recall (horizontally) and precision (vertically) for the three optimization techniques with the Relaxed objective function. Most dots are grouped around the right bottom corner (high recall, low precision) instead of the ideal region towards the top right corner (high recall and precision) where only a few values were found.

7.2.2. Results of MinDiff objective function

Fig. 9b, d and f shows that the recall performance indicator obtains lower values for the MinDiff objective function in all three search based techniques. EA is the best with a value of 0.33. This means that the generated feature models do not include most of the required input feature sets. Using Pearson's correlation coefficient did not reveal any strong correlation between the number of feature sets and recall (all values <−0.18).

Regarding precision, HC (0.66) obtained the best average value followed closely by EA (0.65) and thirdly by RS (0.48). In contrast to the recall results, the precision values are better which means that the feature sets represented by the models are, in most of the cases, part of the input feature sets.

In contrast with the Relaxed function, these results confirm that MinDiff favours precision because it explicitly accounts for surplus feature sets. This is clearly observed in Fig. 9b, d and f which show the values of recall (horizontally) and precision (vertically) for the three optimization techniques and the MinDiff objective function. In these figures, most dots are grouped towards the left top corner (higher precision than recall) instead of the right top one (high recall and precision) where only a few values were found.

7.3. Statistical analysis

The goal of our statistical analysis is twofold. First, to provide formal and quantitative evidences (statistical significance) that the three search based techniques and the objective functions have in fact an impact on the comparison metrics, i.e. that the differences in the results were not obtained by mere chance. Second, to show that those differences are significant in practice (effect size). The statistical analysis of the data was performed using a combination of the R statistical package.[4] and the online statistical tool STATService[5]

7.3.1. Statistical significance

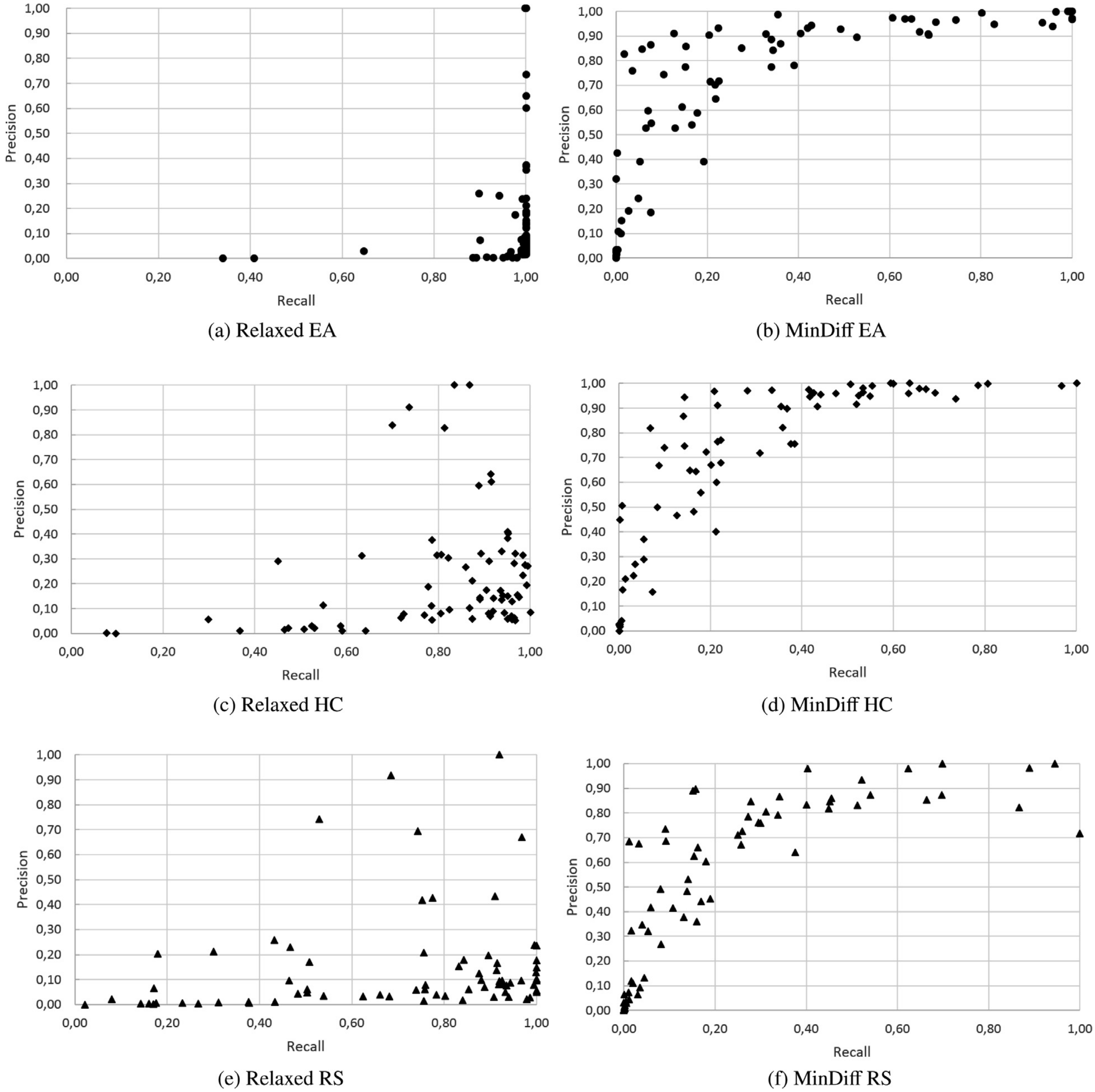
In order to determine whether algorithms have an actual impact on the comparing performance indicators for each objective

[2] Do notice that the 59 models in our conference paper (see Lopez-Herrejon et al., 2012), were selected with *up to* a thousand feature sets. In this paper we include six larger cases to evaluate scalability.

[3] http://www.fosd.de/fh,http://spl2go.cs.ovgu.de/,http://fosd.de/SPLConqueror

[4] http://www.r-project.org/

[5] http://moses.us.es/statservice



(a) Relaxed EA

(b) MinDiff EA

(c) Relaxed HC

(d) MinDiff HC

(e) Relaxed RS

(f) MinDiff RS

Fig. 9. Recall and precision graphs for relaxed and MinDiff objective functions.

function, we used Null Hypothesis Statistical Test (NHST). In NHST, two contrary hypothesis are formulated. The first hypothesis is referred to as *null hypothesis* (H_0) and assumes that the algorithm or objective function has no impact at all on the values of performance indicators, e.g. there is no difference between the recall provided by the algorithms. Opposite to the null hypothesis, an *alternative hypothesis* (H_1) is formulated, stating that the algorithm has a significant effect in the results obtained. Statistical tests provide a probability (called *p-value*) ranging in [0,1]. The lower the *p*-value of a test, the more likely that the null hypothesis is false and the alternative hypothesis is true, e.g. there is a difference in the recall provided by the algorithms. Researchers have established by convention that *p*-values under 0.05 or 0.01 are so-called *statistically significant*, leading to assume that the alternative hypothesis is likely true.

The techniques used to perform the statistical analysis and obtain the *p*-values depend on the statistical properties of the data (Derrac et al., 2011). After some preliminary tests (Kolmogorov–Smirnov, Frank and Massey, 1951 and Shapiro–Wilk, Shapiro and Wilk, 1965 tests) we concluded that our data did not follow a normal distribution in general, thus our analysis requires the use of non–parametric techniques. In particular, since we compare three different algorithms (Derrac et al., 2011), we applied the Friedman test (Derrac et al., 2011). The *p*-values and statistics of such tests are shown in Table 8. Since the *p*-values shown in this table are smaller than 0.01 in all cases, we reject the null hypotheses, and consequently state that there exist differences in the algorithms for all the objective functions and performance indicators (recall and precision).

Table 8
Friedman test statistic and *p*-values.

	Relaxed		MinDiff	
	Recall	Precision	Recall	Precision
p-Value	$\ll$0.01	$\ll$0.01	$\ll$0.01	$\ll$0.01
Statistic	51.9242	50.37	53.3294	50.5696

Table 9
Holm's post hoc *p*-values (and significance threshold).

	Relaxed		MinDiff	
	Recall	Precision	Recall	Precision
EA vs HC	≪0.01	≪0.01	0.4847	0.4349
EA vs RS	≪0.01	0.0096	≪.01	≪0.01
HC vs RS	0.0399	≪0.01	≪0.01	≪0.01

Table 10
$\widehat{A}_{12}$ statistic for each pair of algorithms and combination of objective function and performance measure.

	Relaxed		MinDiff	
	Recall	Precision	Recall	Precision
EA vs HC	0.8950	0.5000	0.5043	0.4742
EA vs RS	0.8689	0.4998	0.5946	0.6565
HC vs RS	0.5822	0.4998	0.6009	0.6612

In order to determine which algorithms are providing such difference, we perform additional post hoc analyses. Those analyses perform pair-wise comparison among the results of each algorithm, determining whether statistically significant differences exist among the results of a specific pair of algorithms. Specifically, Table 9 shows the *p*-values of Holm's post hoc procedure for each pair of algorithm and combination of objective function and performance indicator. The *p*-values shown in this table are smaller (or equal) than its corresponding significance threshold except for the shaded cells. Consequently, we determine that the differences of performance between the algorithms are significant in all cases except for EA and HC for recall and precision using MinDiff as objective function. This similarity can be observed in Fig. 9b and d.

7.3.2. Effect sizes

Even when differences in the performance of the algorithms are statistically significant as in our case, it is also crucial to assess *the magnitude of such a difference*, in order to ensure that such difference has practical value.[6] Following the guidelines provided in (Arcuri and Briand, 2014), we use Vargha and Delaney's $\widehat{A}_{12}$ statistic to evaluate the effect size. Specifically, $\widehat{A}_{12}$ measures the probability that using one algorithm yields higher values than the other for a comparison metric.[7] For example, $\widehat{A}_{12} = 0.8$ entails we would obtain better results 80% of the runs with the first of the algorithms compared, and $\widehat{A}_{12} = 0.2$ entails we would obtain better results 80% of the runs with the other algorithm compared. Thus we have an $\widehat{A}_{12}$ value for each combination of objective function and comparison metric for every pair of algorithms.

Table 10 shows the values of such effect size statistic. We highlight in grey the values with largest distance from 0.5 (value that indicates no difference between algorithms) per column in the table. Results show that the largest difference in recall was between EA and HC with a value of 0.8950, but also a high value (0.8689) was found with RS. This clearly indicates that in terms of recall for the Relaxed function, EA outperforms the other two algorithms. For Relaxed function and precision, the picture is significantly different because there is either no difference between the algorithms (EA vs HC), or the difference is rather negligible (0.4998). This is not surprising as the Relaxed function is not concerned for precision. For the MinDiff function the largest differences were obtained between HC and RS, closely followed by the differences between EA and RS. Overall these results thus confirm that the algorithms used do have an actual impact.

Table 11
Average ranking of Friedman's test for each combination of objective function and algorithm.

Position	Recall			Precision		
	Obj. func.	Alg.	Avg. rank	Obj. func.	Alg.	Avg. rank
#1 (best)	Relaxed	EA	1.1486	MinDiff	HC	2.4595
#2	Relaxed	HC	2.7838	MinDiff	EA	2.6689
#3	F1	EA	3.3919	F1	EA	2.8851
#4	Relaxed	RS	3.4392	F1	HC	3.2500
#5	F1	HC	5.0338	F1	RS	5.3784
#6	F1	RS	6.1824	MinDiff	RS	5.5068
#7	MinDiff	EA	6.9054	Relaxed	LS	6.5405
#8	MinDiff	HC	7.4932	Relaxed	RS	7.9459
#9	MinDiff	RS	8.6216	Relaxed	EA	8.3649

7.4. Reifying trade-off between precision and recall as objective function

The trade-off identified between recall and precision with our two fitness functions (Relaxed and MinDiff) could be captured in different forms. In this paper we express this trade-off using the F_β measure, where β indicates how many times the recall values weight more in comparison with the precision values (Manning et al., 2008). F_β measure is thus defined as follows:

$$F_\beta = \frac{(1+\beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}} \tag{7}$$

From our experience in reverse engineering feature models, recall has usually more importance because software engineers need indeed to make sure they model at least the system variants they currently have to construct a SPL. However, how much more important is recall over precision would vary on a case to case basis depending on factors such as the problem domain or the intended use of the reverse engineered feature models (e.g. for combinatorial interaction testing where low precision may hinder the test effort, Lopez-Herrejon et al., 2014a,b). By giving distinct values to the parameter β the software engineers can define a fitness function that effectively adapts to their concrete needs.

In this paper, we use $\beta = 1$ as an illustrative example where both precision and recall have the same importance. Thus our F_1 measure is defined as

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{8}$$

7.4.1. Statistical analysis of F_1 objective function

Fig. 10 shows the recall and precision graphs obtained for F_1. We performed the same statistical tests as in Section 7.3 for both precision and recall to determine if F_1 improves the results compared with Relaxed and MinDiff.

Table 11 shows the average ranking of Friedman's test for each combination of objective function and algorithm. For recall, F_1 scored 3.3919 when using EA which is only surpassed by the Relaxed function combined with EA or HC. F_1 ranked similarly in precision with a value of 2.8851, but now surpassed by MinDiff combined with HC or EA. These findings show a clear picture, namely, that when recall is more important for the reverse engineering task the Relaxed function wins over F_1 and MinDiff in that order. Complementary, when precision is more important, MinDiff wins over F_1 and Relaxed in that order. In other words, we observe that the Relaxed function performs better for Recall, MinDiff for precision and F_1 provides a trade-off "in between" recall and precision.

[6] Note that with a large enough sample size we can obtain statistically significant differences for distributions with very similar results (Arcuri and Briand, 2014).

[7] We prefer $\widehat{A}_{12}$ to $d = (\mu_A - \mu_B)/\sigma$ (which is the most known and intuitive effect size measure) since it is a non-parametric effect size measure, and consequently it is applicable even when the data does not follow a normal distribution.

Table 12
p-Values of Holms post hoc analyses for recall.

		Relaxed			MinDiff			F1	
		EA	HC	RS	EA	HC	RS	EA	HC
Rel.	HC	0.0003							
	RS	$\ll$0.01	0.1455						
MinDiff	EA	$\ll$0.01	$\ll$0.01	$\ll$0.01					
	HC	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.1917				
	RS	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.0001	0.0122			
F1	EA	$\ll$0.01	0.1768	0.9163	$\ll$0.01	$\ll$0.01	$\ll$0.01		
	HC	$\ll$0.01	$\ll$0.01	0.0004	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.0003	
	RS	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.1083	0.0036	$\ll$0.01	$\ll$0.01	0.0107

Table 13
p-Values of Holms post hoc analyses for precision.

		Relaxed			MinDiff			F1	
		EA	HC	RS	EA	HC	RS	EA	HC
Rel.	HC	$\ll$0.01							
	RS	0.3521	$\ll$0.01						
MinDiff	EA	$\ll$0.01	$\ll$0.01	$\ll$0.01					
	HC	0.0001	$\ll$0.01	$\ll$0.01	0.6418				
	RS	$\ll$0.01	0.0018	0.0217	$\ll$0.01	$\ll$0.01			
F1	EA	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.6311	0.34444	$\ll$0.01		
	HC	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.1968	0.0791	$\ll$0.01	0.4177	
	RS	$\ll$0.01	0.0098	$\ll$0.01	$\ll$0.01	$\ll$0.01	0.7755	$\ll$0.01	$\ll$0.01

Table 14
$\widehat{A}_{12}$ measure for F_1.

			Relaxed			MinDiff			F1	
			EA	HC	RS	EA	HC	RS	HC	RS
Recall	F1	EA	0.1070	0.3672	0.4548	0.7719	0.8296	0.8738	0.6545	0.7528
		HC	0.0367	0.1838	0.3098	0.7012	0.7498	0.8164		0.6398
		RS	0.0362	0.1256	0.2418	0.6298	0.6507	0.7478		
Precision	F1	EA	0.8963	0.8316	0.8893	0.4419	0.4324	0.6026	0.5122	0.6927
		HC	0.8957	0.8285	0.8931	0.4230	0.4133	0.5992		0.6871
		RS	0.8050	0.6954	0.7894	0.2921	0.2850	0.4439		

Tables 12 and 13 show the *p*-values of the permutations between objective functions and algorithms for both precision and recall. It is remarkable that in both tables the number of *p*-values close to zero is high. Concretely, the Holm's procedure rejects those hypothesis having a *p*-value lower than 0.007 for recall and lower than 0.005 for precision. These values support the rejection of the null hypothesis, meaning that the differences in the performance of the algorithms are statistically significant, except for the cells that are highlighted in the tables.

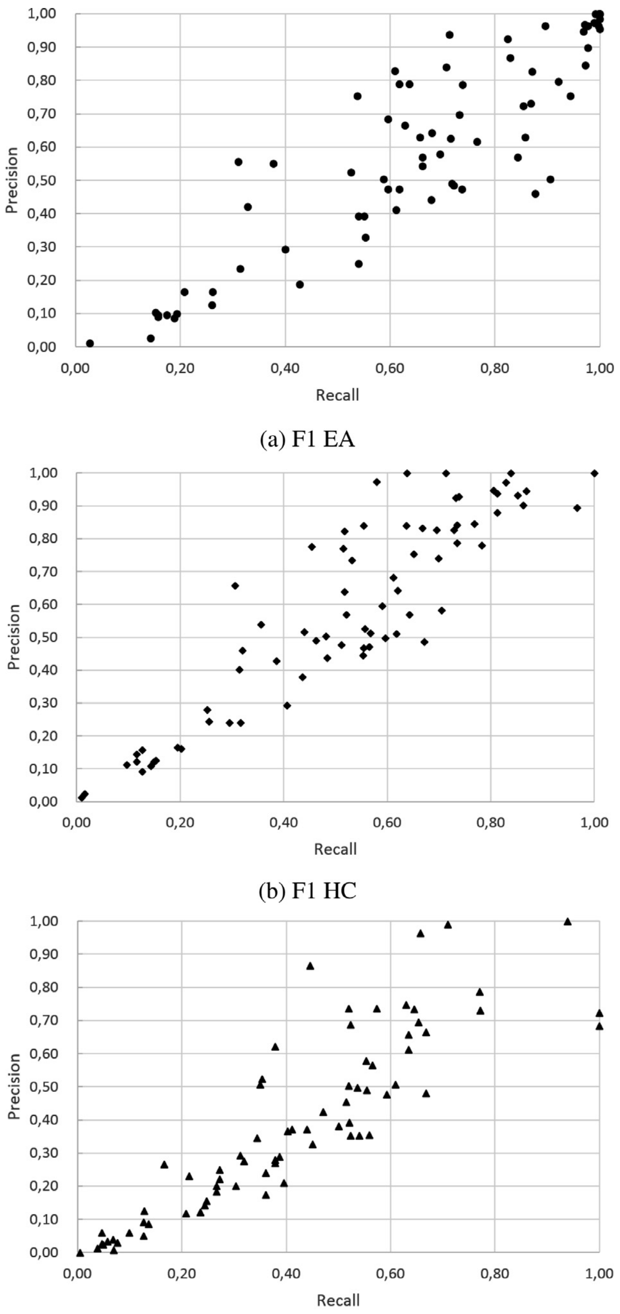
Table 14 shows the $\widehat{A}_{12}$ effect size measures for the F_1 objective function for recall and precision. In terms of recall, the Relaxed objective function clearly outperforms F_1 which again clearly outperforms MinDiff. For precision, F_1 clearly performs better than Relaxed for every algorithm, yet still only slightly worse than MinDiff. The EA and HC algorithms with F_1 perform roughly equally well regarding precision, and both outperform the RS algorithm. For recall EA achieves better results with F_1 as an objective function than HC, but again both outperform RS.

8. Threats to validity

In this section we follow the guidelines suggested by Barros et al. (see de Oliveira Barros and Neto, 2011) to identify the threats to validity that are applicable to our work and describe how they were addressed.

Internal validity threats. Parameter setting is a common internal variability threat. In this paper we used standard values for the search based techniques and values of the domain of feature models (e.g. branching factor). Extensive evidence provided by the work of Arcuri and Fraser (2013) suggests that default values might be good enough for assessing some search based techniques in the context of testing. However, how their findings map to our particular context is an open question that we plan to address in the future work. Throughout the paper we discuss and illustrate how our code was implemented and how the data was gathered and analysed. The source code and related artefacts are available for replication. We should also point out that the case studies we used to generate the input feature sets are publicly available to the research community and aim to represent realistic examples of product lines.

External validity threats. We identified two external threats. The first threat is the selection of the feature models corpus. We selected a group of feature models that have a realistic number of products, meaning that companies that employ SPL practices can commonly support them. Certainly other case studies could be considered that would be worth including and might yield different results. Our ultimate target is to analyse the so-called *feature models in the wild* (Berger, 2012). This set of case studies are perhaps the largest and most complex collection of feature models, most of them are from the operating systems domain. Coping with their level of complexity does demand significant tool improvement beyond the scope of the current work. For instance, improvements are needed to



(a) F1 EA

(b) F1 HC

(c) F1 RS

Fig. 10. Recall and precision graphs for the F1 objective function.

effectively represent larger number of features and feature sets to efficiently implement the required operators in combination with the underlying feature modelling reasoning tools.

The second threat to validity is the selection of search based algorithms. In this paper we chose three basic algorithms that are commonly used in empirical research in Search Based Software Engineering. Certainly, different choices of algorithms and corresponding parameter settings could yield different results. We want to explore other possibilities and settings in our future work.

Construct validity threats. In our work we identified two threats of validity of this kind. The first threat is the *lack of assessing the validity of cost measures* (de Oliveira Barros and Neto, 2011). To address this threat, we made sure that a fair comparison was made among the three search algorithms by using the same number of objective function evaluations. The second threat is *lack of assessing the validity of effectiveness measures* (de Oliveira Barros and Neto, 2011). This threat is addressed because we used standard information retrieval metrics to assess the quality of the obtained results.

We do acknowledge that our fitness functions resemble the definitions of recall and precision. RelaxedFF can be regarded as a non-normalized form of recall. MinDiffFF is defined in terms of deficit and surplus which in turn resemble the definitions of recall and precision respectively, expressed in terms of differences of set sizes rather than ratios. Nonetheless, we argue, based on our experience, that these two objective functions do indeed capture the most common scenarios for reverse engineering feature models.

Conclusion validity threats. To address this type of threats in our work we: (i) considered 30 independent runs for each combination of feature model with search algorithm and objective function, (ii) employed standard statistical analysis following accepted guidelines (Arcuri and Briand, 2014), and (iii) used random search as a standard comparison baseline.

9. Related work

There is extensive and increasing literature in reverse engineering feature models, especially from source code artefacts. In this section, we summarize those pieces of work closest to ours. In addition, we shortly summarize two recent articles about the application of search based techniques to SPLs.

The work by Haslinger et al. (2011) proposes an ad hoc algorithm to reverse engineer feature models from feature sets. It works by identifying occurrence patterns in the selected and not selected features that are mapped to parent–child relations of feature models. This work has been extended to consider requires and excludes CTCs (Haslinger et al., 2013); however, it does not support more general types of CTCs. The main distinction with our work is that only one feature model can be reversed engineered, whereas in our approach we could provide different feature models (if they exist) as alternatives for the designers to choose from. Nonetheless, we believe Haslinger et al.'s algorithms could be used to seed the search in our techniques. This is an issue we plan to explore in our future work. Our recent work has explored using genetic programming for reverse engineering feature models (Linsbauer et al., 2014). A detailed comparison with this approach is also part of our future work.

The work by Czarnecki and Wasowski (2007) studies reverse engineering of feature models but from a set of propositional logic formulas. They provide an ad hoc algorithm that can potentially extract from a single propositional logic formula multiple feature models but that tries to preserve the original formulas and reduce redundancies. Subsequent work by She et al. highlighted the limitations of this approach, namely problems selecting the parents of features and incompleteness (She et al., 2011). They proposed a heuristic to address these two issues that complements dependency information with textual feature description. A recent extension of this work provides improved algorithms based on CNF and DNF constraints (She et al., 2014). In contrast with our work, their starting point are configuration files, documentation files, and constraints expressed in propositional logic.

Yi et al. (2012) use classifiers to identify binary constraints but do not reach the point of constructing a feature model as proposed by our work. Nevertheless, we believe their work could also be used to seed the search algorithms. We aim to address this issue as part of our future work. Closer to our work is Acher et al. (2012) that also tackle the reverse engineering of feature models from feature sets. The salient difference between our approaches is that their work maps each feature set into a feature model which are later merged in to a single feature model. Also citetmathieu13, proposed to reengineer plugin-based architectures to obtain a variability model describing the dependencies existing on it. The authors propose to generate a model to describe the architecture of the system and another model to describe the dependencies. These two models are consequently merged to represent the variability of the plugin system. The mapping and merge operations rely on propositional logic techniques and tools which can be computationally expensive. A more detailed comparison and analysis of the advantages and disadvantages of both approaches is part of our future work.

Sannier et al. performed an analysis of matrices for products comparison available at Wikipedia. Their work identifies variability patterns in the values of the matrix cells, and portrays the challenges and potential benefits of exploiting that information, among other things, to extract models such as feature models (Sannier et al., 2013). One difference with our work is that their product matrices can contain other values, rather than selected or not selected as in our case. In addition, to the best of our knowledge, at present there is no algorithm let alone tool support for this approach.

Davril et al. (2013) address the problem of incomplete and informal product descriptions. In contrast with our work, the authors do not assume the existence of a table of system variants like Table 1. Instead, their focus is on harvesting the product information from website product descriptions. To the harvested information, they apply ad hoc algorithms to reconstruct a feature model. We believe this work is complementary to ours as it could be used to obtain further and ideally more realistic case studies. A more detailed comparison on the obtained feature models with both approaches is also part of our future work.

Harman et al. (2014) performed a survey on the topic of SBSE applied to SPLs. They present an overview of recent articles classified according to themes such as configuration, testing, or architectural improvement. Most salient is their proposal on using Genetic Improvement, a variation of Genetic Programming (Poli et al., 2008), whose goal is to improve existing programs rather than creating them from scratch. Lopez-Herrejon et al. (2014c) performed a preliminary systematic mapping study at the connection of SBSE and SPL. In contrast with Harman et al.'s work, they categorized the articles along a known framework for SPL development. Furthermore, this mapping study also considered the types of case studies used, their number and provenance, as well as how they were empirically analyzed. Both studies pointed out the prevalence of testing as the main application of SBSE techniques and highlighted the increasing number of publications on the subject in recent years.

10. Future work

We argue that our study has opened up several research venues on the application of SBSE techniques for variability management. The following are some areas we plan to pursue as future work.

Parameter landscape analysis. Our three search based techniques were initialized with standard parameter values and default domain values (e.g. feature model branching factor). Thus an analysis of the parameter landscape of the problem is duly called for. The ultimate goal is to see whether any particular parameter configurations can yield better results and how they would scale for larger feature models and feature sets.

Advanced search-based techniques. Our current work explored three basic search algorithms with standard and basic settings. We plan to assess other possibilities, for instance, different selection strategies beyond roulette wheel.

Improvement of objective functions. The cornerstone of our work is devising an adequate objective function that contains the set of products required, as tight as possible, but still remains scalable. A possibility is to experiment with functions that consider feature model metrics (e.g. Bagheri and Gasevic, 2011) or perhaps exploiting domain knowledge, such as feature hierarchies, that can help to trim the search space. We also plan to explore performance improvements for the feature model operations that the objective functions rely on. The work of Thüm et al. (2009) and Mendonça et al. (2009) are our starting points.

Variability-aware mutation operators. Currently our work applies mutation operators without any considerations of any potential variability implications, that is, how it could impact the set of products denoted by a feature model. We want to extend the set of mutation operators so that they consider the impact they may have on variability. We could then set up different probabilities so that they could be applied distinctly perhaps depending on the nature of the required set of products. Integrating the work on analysis of feature model changes is a starting point (Thüm et al., 2009; Segura et al., 2011).

Quality of reverse-engineered feature models. So far the emphasis of our work has been on obtaining a feature model that denotes the required set of feature sets. However, as we mentioned, more than one feature model can denote the same set of feature sets. The question is now, towards which equivalent feature model should the search be directed? We believe that quality metrics for feature models (Bagheri and Gasevic, 2011) as well as quantification of developers feedback could also be integrated (Acher et al., 2011) to help answer this question.

Exploiting genetic programming. Feature models can have more complex relations among features, e.g. arbitrary CTCs, and can be provided with more attribute values (Benavides et al., 2010). These two characteristics may be better addressed with genetic programming (Poli et al., 2008), which provides more flexible alternatives to manipulate the tree-like structures of the feature models. Our recent work on the subject is but a first step in this direction (Linsbauer et al., 2014). We believe genetic programming could prove useful not only for our reverse engineering goals but also for other related problems of variability management.

Comparative studies. As mentioned throughout the paper, specially in the related work, we plan to perform a detailed comparative study of other reverse engineering approaches to assess both their quality and their complementarity to our work (Haslinger et al., 2011; Yi et al., 2012; Acher et al., 2012, 2013; Davril et al., 2013).

Multi-objective optimization formulation. Our work highlighted a clear trade-off between precision and recall that we reified into a third fitness function based on F_1 measure with good performance results. This also opens up the possibility of exploring our reverse engineering tasks as multi-objective optimization problem (Coello et al., 2007; Deb, 2001). In our case, precision, recall or other performance metrics can be the objectives to optimize depending on the particular needs of the concrete problem domains. As a first step, we will explore classical multi-objective algorithms such as NSGA-II and SPEA2.

11. Conclusions

As SPLs are becoming a more pervasive development paradigm so is the need to reverse engineer feature models to capture their variability. In this paper we make an assessment of three search based techniques using three objective functions to reverse engineer feature models from the feature sets that describe system variants of a SPL.

We compared and contrasted the combinations of Relaxed and MinDiff for three search algorithms. The results showed a clear trade-off between recall and precision for these objective functions. Relaxed captures the scenario where recall is more important for the reverse engineering task and obtained the best results using the evolutionary algorithm. MinDiff, on the other hand, captures the scenario where it is also important to have a good precision (e.g. for software testing purposes). Here the results are slightly better for hill climbing than for the evolutionary algorithm but not statistically significant. Furthermore, we captured the trade-off between recall and precision using F_β measure and evaluated as a third objective function F_1, which equally favours recall and precision. For F_1 the best results where obtained by the evolutionary algorithm. We argue that using F_β measure as an objective function gives software engineers the flexibility to weigh in different relationships between recall and precision applicable to the concrete problem domains of the reverse engineering task.

Our work also helped to identify several research venues that we plan to address as future work. We believe that the work presented in this paper is a stepping stone towards leveraging the wealth of Search-Based Software Engineering techniques not only for reverse engineering feature models but also to address many variability management challenges. The sources and data can be downloaded from: http://www.sea.jku.at/tools/jss-sbse.

Acknowledgments

This research is partially funded by the Austrian Science Fund (FWF) project P 25289-N15 and Lise Meitner Fellowship M1421-N15, and Marie Curie Actions – Intra-European Fellowship (IEF) project number 254965. This work has been partially supported by the European Commission (FEDER), by the Spanish Government under TAPAS (TIN2012-32273) project and by the Andalusian Government under the Talentia program, and COPAS project (TIC-1867). We thank Michael Affenzeller and his team for their support with HeuristicLab for the earlier implementation of our work.

Appendix A. Repairing strategy in ETHOM

In ETHOM, we identified three types of patterns making a chromosome infeasible or semantically redundant, namely: (i) those encoding set relationships (or- and alternative) with a single child feature (Fig. 11a), (ii) those containing cross-tree constraints between features with parental relationship (Fig. 11b), and (iii) those containing features linked by contradictory or redundant cross-tree constraints (Fig. 11c). To remove these inconsistencies we applied the following repair algorithm: (i) isolated set relationships are converted into optional relationships (e.g. the model in Fig. 11a is changed as in Fig. 11d), (ii) cross-tree constraints between features with parental relationships are removed (e.g. the model in Fig. 11b is changed as in Fig. 11e), and (iii) two features cannot be linked by more than one cross-tree constraint (e.g. the model in Fig. 11c is changed as in Fig. 11f). For further details, please refer to (Segura et al., 2014).

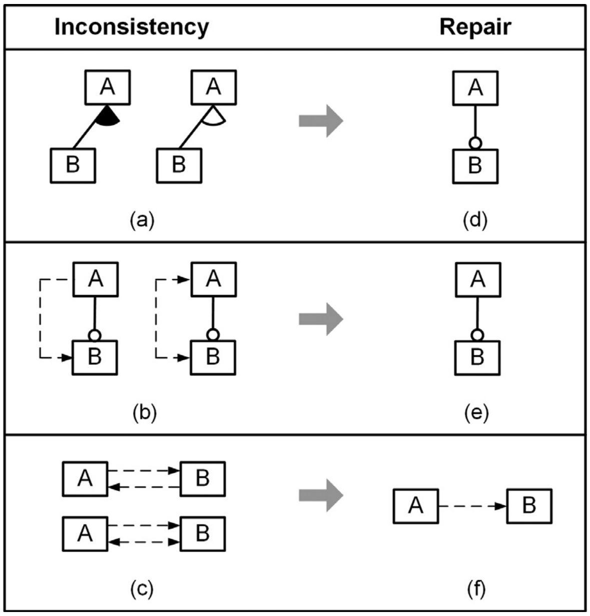


Fig. 11. Examples of infeasible individuals and repairs.

References

Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P., 2011. Reverse engineering architectural feature models. In: Crnkovic, I., Gruhn, V., Book, M. (Eds.), ECSA, Vol. 6903 of Lecture Notes in Computer Science. Springer, pp. 220–235.

Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P., 2012. On extracting feature models from product descriptions. In: Eisenecker, U.W., Apel, S., Gnesi, S. (Eds.), Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25–27, 2012. Proceedings, ACM, pp. 45–54..

Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P., 2013. Extraction and evolution of architectural variability models in plugin-based systems. Softw. Syst. Model., 1–28. doi:10.1007/s10270-013-0364-2.

Arcuri, A., Briand, L.C., 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab. 24 (3), 219–250.

Arcuri, A., Fraser, G., 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. Empir. Softw. Eng. 18 (3), 594–623.

Bagheri, E., Gasevic, D., 2011. Assessing the maintainability of software product line feature models using structural metrics. Softw. Qual. J. 19 (3), 579–612.

Benavides, D., Segura, S., Trinidad, P., Cortés, A.R., 2007. Fama: tooling a framework for the automated analysis of feature models. In: Pohl, K., Heymans, P., Kang, K.C., Metzger, A. (Eds.), VaMoS, Vol. 2007-01 of Lero Technical Report, pp. 129–134.

Benavides, D., Segura, S., Cortés, A.R., 2010. Automated analysis of feature models 20 years later: a literature review. Inf. Syst. 35 (6), 615–636.

Benavides, D., Trinidad, P., Cortés, A.R., 2013. Automated reasoning on feature models. In: Bubenko Jr., J.A., Krogstie, J., Pastor, O., Pernici, B., Rolland, C., Sølvberg, A. (Eds.), Seminal Contributions to Information Systems Engineering. Springer, pp. 361–373.

Berger, T., 2012. Variability modeling in the wild. In: de Almeida, E.S., Schwanninger, C., Benavides, D. (Eds.), SPLC (2). ACM, pp. 233–241.

Coello, C.C., Lamont, G.B., Veldhuizen, D.A., 2007. Evolutionary algorithms for solving multi-objective problems, In: Genetic and Evolutionary Computation, second ed. Springer.

Czarnecki, K., Eisenecker, U., 2000. Generative Programming: Methods, Tools, and Applications. Addison-Wesley.

Czarnecki, K., Wasowski, A., 2007. Feature diagrams and logics: there and back again, SPLC. IEEE Computer Society, pp. 23–34.

Davril, J.-M., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., Heymans, P., 2013. Feature model extraction from large collections of informal product descriptions, In: Proceedings of the 2013 Ninth Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013. ACM, New York, NY, USA, pp. 290–300. doi:10.1145/2491411.2491455.

de Oliveira Barros, M., Neto, A.C.D., 2011. Threats to Validity in Search-based Software Engineering Empirical Studies, Tech. Rep. 0006/2011. Universidade Federal Do Estado Do Rio de Janeiro, Departamento de Informatica Aplicada..

Deb, K., 2001. Multi-Objective Optimization Using Evolutionary Algorithms, first ed. Wiley.

Derrac, J., Garcìa, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evolut. Comput. 1 (1), 3–18. doi:10.1016/j.swevo.2011.02.002 http://www.sciencedirect.com/science/article/pii/S2210650211000034.

Eiben, A., Smith, J., 2003. Introduction to Evolutionary Computing. Springer-Verlag.

Frank, J., Massey, J., 1951. The Kolmogorov–Smirnov test for goodness of fit. J. Am. Stat. Assoc. 46 (253), 68–78.

Generative Software Development Lab, C., 2013. Computer Systems Group, University of Waterloo, Software Product Line Online Tools (SPLOT) http://www.splot-research.org/.

Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A., 2011. Reverse engineering feature models from programs' feature sets. In: Pinzger, M., Poshyvanyk, D., Buckley, J. (Eds.), WCRE. IEEE Computer Society, pp. 308–312.

Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J., Zhang, Y., 2014. Search based software engineering for software product line engineering: a survey and directions for future work. In: Gnesi, S., Fantechi, A., Heymans, P., Rubin, J., Czarnecki, K. (Eds.), SPLC. ACM, pp. 5–18.

Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A., 2013. On extracting feature models from sets of valid feature combinations. In: Cortellessa, V., Varró, D. (Eds.), FASE, Vol. 7793 of Lecture Notes in Computer Science. Springer, pp. 53–67.

Hetrick, W.A., Krueger, C.W., Moore, J.G., 2006. Incremental return on incremental investment: Engenio's transition to software product line practice. In: Tarr, P.L., Cook, W.R. (Eds.), OOPSLA Companion. ACM, pp. 798–804.

Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A., 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. Rep. CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University.

Krueger, C.W., 2001. Easing the transition to software mass customization. In: van der Linden, F. (Ed.), PFE, vol. 2290 of Lecture Notes in Computer Science. Springer, pp. 282–293.

Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A., 2014. Feature model synthesis with genetic-programming. In: Le Goues, C., Yoo, S. (Eds.), SSBSE, Vol. 8636 of Lecture Notes in Computer Science. Springer, pp. 153–167.

Lopez-Herrejon, R.E., Galindo, J.A., Benavides, D., Segura, S., Egyed, A., 2012. Reverse engineering feature models with evolutionary algorithms: an exploratory study. In: Fraser, G., de Souza, J.T. (Eds.), SSBSE, Vol. 7515 of Lecture Notes in Computer Science. Springer, pp. 168–182.

Lopez-Herrejon, R.E., Ferrer, J., Chicano, J.F., Haslinger, E.N., Egyed, A., Alba, E., 2014a. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In: Arnold, D.V. (Ed.), GECCO. ACM, pp. 1255–1262.

Lopez-Herrejon, R.E., Ferrer, J., Chicano, J.F., Egyed, A., Alba, E., 2014b. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines, CEC. IEEE, pp. 387–396.

Lopez-Herrejon, R.E., Ferrer, J., Chicano, F., Linsbauer, L., Egyed, A., Alba, E., 2014. A Hitchhiker's guide to search-based software engineering for software product lines. CoRR. (abs/1406.2823) http://arxiv.org/abs/1406.2823 .

Luke, S., 2009. Essentials of Metaheuristics, Lulu Available at: http://cs.gmu.edu/~sean/book/metaheuristics/.

Manning, C.D., Raghavan, P., Schütze, H., 2008. Introduction to Information Retrieval. Cambridge University Press.

Mendonça, M., Wasowski, A., Czarnecki, K., 2009. Sat-based analysis of feature models is easy. In: Muthig, D., McGregor, J.D. (Eds.), Software Product Lines, 13th International Conference, SPLC 2009. Proceedings, Vol. 446 of ACM International Conference Proceeding Series, San Francisco, CA, USA, August 24–28, 2009. ACM, pp. 231–240.

Pohl, K., Bockle, G., van der Linden, F.J., 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer.

Poli, R., Langdon, W.B., McPhee, N.F., 2008. A Field Guide to Genetic Programming. Lulu.

Sannier, N., Acher, M., Baudry, B., 2013. From comparison matrix to variability model: the wikipedia case study, ASE. IEEE, pp. 580–585.

Segura, S., Hierons, R.M., Benavides, D., Cortés, A.R., 2011. Automated metamorphic testing on the analyses of feature models. Inf. Softw. Technol. 53 (3), 245–258.

Segura, S., Galindo, J., Benavides, D., Parejo, J.A., Cortés, A.R., 2012. BeTTy: benchmarking and testing on the automated analysis of feature models. In: Eisenecker, U.W., Apel, S., Gnesi, S. (Eds.), Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25–27, 2012. Proceedings, ACM, pp. 63–71..

Segura, S., Parejo, J.A., Hierons, R.M., Benavides, D., Cortés, A.R., 2014. Automated generation of computationally hard feature models using evolutionary algorithms. Expert Syst. Appl. 41 (8), 3975–3992.

Shapiro, S.S., Wilk, M.B., 1965. An analysis of variance test for normality. Biometrika 52 (3–4), 591–611.

She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K., 2011. Reverse engineering feature models. In: Taylor, R.N., Gall, H., Medvidovic, N. (Eds.), ICSE. ACM, pp. 461–470.

She, S., Ryssel, U., Andersen, N., Wasowski, A., Czarnecki, K., 2014. Efficient synthesis of feature models. Inf. Softw. Technol. 56 (9), 1122–1143.

Svahnberg, M., van Gurp, J., Bosch, J., 2005. A taxonomy of variability realization techniques. Softw. Pract. Exp. 35 (8), 705–754.

Thüm, T., Batory, D.S., Kästner, C., 2009. Reasoning about edits to feature models, ICSE. IEEE, pp. 254–264.

van d. Linden, F.J., Schmid, K., Rommes, E., 2007. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer.

Weston, N., Chitchyan, R., Rashid, A., 2009. A framework for constructing semantically composable feature models from natural language requirements. In: Muthig, D., McGregor, J.D. (Eds.), Software Product Lines, 13th International Conference, SPLC 2009. Proceedings, Vol. 446 of ACM International Conference Proceeding Series, San Francisco, CA, USA, August 24–28, 2009. ACM, pp. 211–220.

Yi, L., Zhang, W., Zhao, H., Jin, Z., Mei, H., 2012. Mining binary constraints in the construction of feature models, Requirements Engineering Conference (RE), 2012 20th IEEE International, pp. 141–150. doi:10.1109/RE.2012.6345798.

Zave, P., Faq sheet on feature interaction, http://www.research.att.com/pamela/faq.html.

Dr. Roberto Erick Lopez-Herrejon is currently a senior postdoctoral researcher at the Johannes Kepler University in Linz, Austria. He was Lise Meitner Fellow (2012–2014) at the same institution. From 2008 to 2014 he was an External Lecturer at the Software Engineering Masters Programme of the University of Oxford, England. From 2010 to 2012 he held an FP7 Intra-European Marie Curie Fellowship on a project for consistency and composition of variable systems with multi-view models. He obtained his Ph.D. from the University of Texas at Austin in 2006, funded in part by a Fulbright Fellowship sponsored by the U.S. State Department. From 2005 to 2008, he was a Career Development Fellow at the Software Engineering Centre of the University of Oxford sponsored by Higher Education Founding Council of England (HEFCE). His expertise is software product lines, variability management, feature oriented software development, model driven software engineering, and consistency checking.

Lukas Linsbauer is a Ph.D. student at the Johannes Kepler University in Linz, Austria. He recently obtained his M.Sc. from the same institution in the area of recovering traceability to reverse engineer software product lines.

Jose Antonio Galindo is currently a Ph.D. student at the University of Seville, Spain. His area of interest is feature model analysis.

Jose Antonio Parejo has recently obtained his Ph.D. from the University of Seville, Spain. His area of interest is evolutionary computation.

David Benavides is an Associate Professor at the University of Seville, Spain. His main area of expertise is Software Product Lines. He is one of the initial developers of FAMA, the most widely used tool for the analysis of feature models.

Sergio Segura is a Lecturer at the Department of Computer Languages and Systems at the University of Seville. He obtained his Ph.D. (with honors) from that institution in 2011. He is a member of the Applied Software Engineering research group. His current research interests include: software product lines, AI techniques in software engineering, automated treatment of variability, AI techniques in software testing, testing and benchmarking of FM analysis tools, model-based testing, testing of variability-intensive systems, testing automation.

Prof. Dr. Alexander Egyed heads the Institute for Software Engineering and Automation at the Johannes Kepler University, Austria. He is also an Adjunct Assistant Professor at the University of Southern California, USA. Before joining the JKU, he worked as a Research Scientist for Teknowledge Corporation, USA (2000–2007) and then as a Research Fellow at the University College London, UK (2007–2008). He received a Doctorate degree in 2000 and a Master of Science degree in 1996, both in Computer Science, from the University of Southern California, USA under the mentorship of Dr. Barry Boehm. His research interests include software design modeling, requirements engineering, consistency checking and resolution, traceability, and change impact analysis. He is a member of ACM, ACM SigSoft, IEEE, and IEEE Computer Society.