The Journal of Systems and Software xxx (2014) xxx–xxx

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
Received in revised form 10 July 2014
Accepted 14 October 2014
Available online xxx

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/

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.

Please cite this article in press as: Lopez-Herrejon, R.E., et al., An assessment of search-based techniques for reverse engineering feature models. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.10.037