Expert Systems with Applications 41 (2014) 3975–3992

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Automated generation of computationally hard feature models using evolutionary algorithms

Sergio Segura [a,1], José A. Parejo [a,*], Robert M. Hierons [b], David Benavides [a], Antonio Ruiz-Cortés [a]

[a] Department of Computer Languages and Systems, University of Seville, Av Reina Mercedes S/N, 41012 Seville, Spain
[b] School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex UB7 7NU, United Kingdom

ARTICLE INFO

Keywords:
Search-based testing
Software product lines
Evolutionary algorithms
Feature models
Performance testing
Automated analysis

ABSTRACT

A feature model is a compact representation of the products of a software product line. The automated extraction of information from feature models is a thriving topic involving numerous analysis operations, techniques and tools. Performance evaluations in this domain mainly rely on the use of random feature models. However, these only provide a rough idea of the behaviour of the tools with average problems and are not sufficient to reveal their real strengths and weaknesses. In this article, we propose to model the problem of finding computationally hard feature models as an optimization problem and we solve it using a novel evolutionary algorithm for optimized feature models (ETHOM). Given a tool and an analysis operation, ETHOM generates input models of a predefined size maximizing aspects such as the execution time or the memory consumption of the tool when performing the operation over the model. This allows users and developers to know the performance of tools in pessimistic cases providing a better idea of their real power and revealing performance bugs. Experiments using ETHOM on a number of analyses and tools have successfully identified models producing much longer executions times and higher memory consumption than those obtained with random models of identical or even larger size.

1. Introduction

Software Product Line (SPL) engineering is a systematic reuse strategy for developing families of related software systems (Clements & Northrop, 2001). The emphasis is on deriving products from a common set of reusable assets and, in doing so, reducing production costs and time-to-market. The products of an SPL are defined in terms of features where a *feature* is any increment in product functionality (Batory, 2005). An SPL captures the commonalities (i.e., common features) and variabilities (i.e., variant features) of the systems that belong to the product line. This is commonly done by using a so-called feature model. A *feature model* (Kang, Cohen, Hess, Novak, & Peterson, 1990) represents the products of an SPL in terms of features and relationships amongst them (see the example in Fig. 1).

The automated extraction of information from feature models (a.k.a. automated analysis of feature models) is a thriving topic that has received much attention in the last two decades (Benavides, Segura, & Ruiz-Cortés, 2010). Typical analysis operations allow us to know whether a feature model is consistent (i.e., it represents at least one product), the number of products represented by a feature model, or whether a model contains any errors. Catalogues with up to 30 analysis operations on feature models have been reported (Benavides et al., 2010). Techniques that perform these operations are typically based on propositional logic (Batory, 2005; Mendonca, Wasowski, & Czarnecki, 2009), constraint programming (Benavides, Ruiz-Cortés, & Trinidad, 2005; White, Schmidt, Benavides, Trinidad, & Ruiz-Cortés, 2008), or description logic (Wang, Li, Sun, Zhang, & Pan, 2007). Also, these analysis capabilities can be found in several commercial and open source tools including AHEAD Tool Suite (2013), Biglever software gears (2013), FaMa Framework (2013), Feature Model Plug-in (2013), pure::variants (2013) and SPLOT (Mendonca, Branco, & Cowan (2009)).

The development of tools and benchmarks to evaluate the performance and scalability of feature model analysis tools has been recognised as a challenge (Batory, Benavides, & Ruiz-Cortés, 2006; Benavides et al., 2010; Pohl, Lauenroth, & Pohl, 2011; She, Lotufo, Berger, Wasowski, & Czarnecki, 2011). Also, recent publications reflect an increasing interest in evaluating and comparing the performance of techniques and tools for the analysis of feature models (Andersen, Czarnecki, She, & Wasowski, 2012; Henard, Papadakis, Perrouin, Klein, & Traon, 2013; Heradio-Gil, Fernandez-Amoros, Cerrada, & Cerrada, 2011; Johansen, Haugen, & Fleurey, 2012; Mendonca et al., 2009; Lopez-Herrejon, Chicano, Ferrer, Egyed, & Alba, 2013; Perrouin et al., 2012; Pohl et al., 2011; Pohl, Stricker, & Pohl, 2013; Sayyad, Menzies, & Ammar,

* Corresponding author. Tel.: +34 954 556 881; fax: +34 954 557 139.
E-mail addresses: sergiosegura@us.es (S. Segura), japarejo@us.es (J.A. Parejo).
[1] Principal corresponding author.

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.12.028

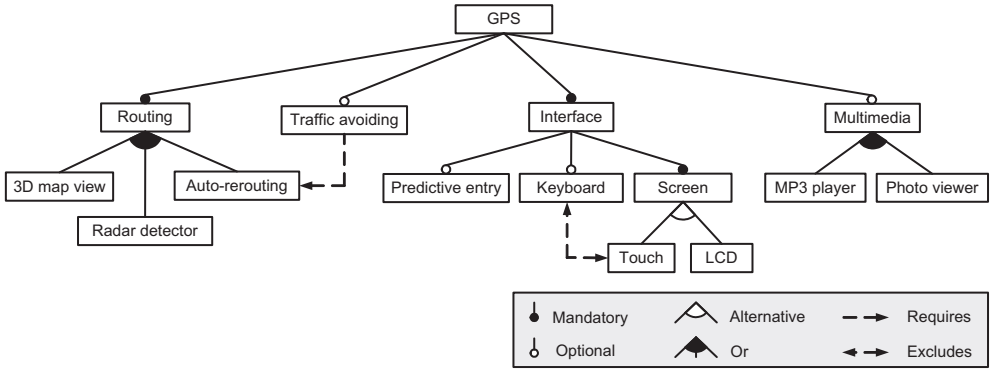

Fig. 1. A sample feature model.

2013; Soltani, Asadi, Gasevic, Hatala, & Bagheri, 2012; Wang, Ali, & Gotlieb, 2013). One of the main challenges when performing experiments is finding tough problems that show the strengths and weaknesses of the tools under evaluation in extreme situations, e.g., those producing longest execution times. Feature models from real domains are by far the most appealing input problems. Unfortunately, although there are references to real feature models with hundreds or even thousands of features (Batory et al., 2006; Loesch & Ploedereder, 2007; Steger et al., 2004), only portions of them are usually available. This lack of hard realistic feature models has led authors to evaluate their tools with large randomly generated feature models of 5000 (Mendonca, Wasowski, Czarnecki, & Cowan, 2008; White et al., 2008), 10,000 (Guo, White, Wang, Li, & Wang, 2011; Mendonca et al., 2009; Thüm, Batory, & Kästner, 2009; White, Doughtery, & Schmidt, 2009) and up to 20,000 (Osman, Phon-Amnuaisuk, & Ho, 2009) features. In fact, the size of the feature models used in experiments has been increasing, suggesting that authors are looking for complex problems on which to evaluate their tools (Benavides et al., 2010). More recently, some authors have suggested looking for hard and realistic feature models in the open source community (Berger, She, Lotufo, Wasowski, & Czarnecki, 2010; Galindo, Benavides, & Segura, 2010; Passos et al., 2011; She, Lotufo, Berger, Wasowski, & Czarnecki, 2010; She et al., 2011). For instance, She et al. (2011) extracted a feature model containing more than 5000 features from the Linux kernel.

The problem of generating test data to evaluate the performance of software systems has been largely studied in the field of software testing. In this context, researchers realised long ago that random values are not effective in revealing the vulnerabilities of a system under test. As pointed out by McMinn (2004): *"random methods are unreliable and unlikely to exercise 'deeper' features of software that are not exercised by mere chance"*. In this context, metaheuristic search techniques have proved to be a promising solution for the automated generation of test data for both functional (McMinn, 2004) and non–functional properties (Afzal, Torkar, & Feldt, 2009). *Metaheuristic search techniques* are frameworks which use heuristics to find solutions to hard problems at an affordable computational cost. Examples of metaheuristic techniques include evolutionary algorithms, hill climbing, and simulated annealing (Voß, 2001). For the generation of test data, these strategies translate the test criterion into an objective function (also called a fitness function) that is used to evaluate and compare the candidate solutions with respect to the overall search goal. Using this information, the search is guided toward promising areas of the search space. Wegener, Grimm, Grochtmann, and Sthamer (1996) and Wegener, Sthamer, Jones, and Eyres (1997) were one of the first to propose the use of evolutionary algorithms to verify the time constraints of software back in 1996. In their work, the authors used genetic algorithms to find input combinations that violate the time constraints of real-time systems, that is, those inputs producing an output too early or too late. Their experimental results showed that evolutionary algorithms are much more effective than random search in finding input combinations maximising or minimising execution times. Since then, a number of authors have followed their steps using metaheuristics and especially evolutionary algorithms for testing non-functional properties such as execution time, quality of service, security, usability or safety (Afzal et al., 2009; McMinn, 2004).

1.1. Problem description

Current performance evaluations on the analysis of feature models are mainly carried out using randomly generated feature models. However, these only provide a rough idea of the average performance of tools and do not reveal their specific weak points. Thus, the SPL community lacks mechanisms that take analysis tools to their limits and reveal their real potential in terms of performance. This problem has negative implications for both tool users and developers. On the one hand, tool developers have no means of performing exhaustive evaluations of the strengths and weaknesses of their tools making it hard to find faults affecting their performance. On the other hand, users are not provided with full information about the performance of tools in pessimistic cases and this makes it difficult for them to choose the tool that best meets their needs. Hence, for instance, a user could choose a tool based on its average performance and later realise that it performs very badly in particular cases that appear frequently in their application domain.

In this article, we address the problem of generating computationally hard feature models as a means to reveal the performance strengths and weaknesses of feature model analysis tools. The problem of generating hard feature models has traditionally been addressed by the SPL community by simply randomly generating huge feature models with thousands of features and constraints. That is, it is generally observed and assumed that the larger the model the harder its analysis. However, we remark that these models are still randomly generated and therefore, as warned by software testing experts, they are not sufficient to exercise the specific features of a tool under evaluation. Another negative consequence of using huge feature models to evaluate the performance of tools is that they frequently fall out of the scope of their users. Hence, both developers and users would probably be more interested in knowing whether a tool may crash with a hard model of small or medium size.

Finally, we may mention that using realistic or standard collections of problems (i.e., benchmarks) is equally insufficient for an