Exploiting the Enumeration of All Feature Model Configurations

A New Perspective with Distributed Computing

José A. Galindo
Inria, Rennes, France
jagalindo@inria.fr

Mathieu Acher
University of Rennes 1 and Inria, France
mathieu.acher@inria.fr

Juan Manuel Tirado
Cambridge Computer Lab
jmt78@cl.cam.ac.uk

Cristian Vidal
Universidad Autónoma de Chile, Talca, Chile
cristian.vidal@uautonoma.cl

Benoit Baudry
Inria, Rennes, France
benoit.baudry@inria.fr

David Benavides
University of Seville, Spain
benavides@us.es

ABSTRACT

Feature models are widely used to encode the configurations of a software product line in terms of mandatory, optional and exclusive features as well as propositional constraints over the features. Numerous computationally expensive procedures have been developed to model check, test, configure, debug, or compute relevant information of feature models. In this paper we explore the possible improvement of relying on the enumeration of all configurations when performing automated analysis operations. The key idea is to precompile configurations so that reasoning operations (queries and transformations) can then be performed in polytime. We tackle the challenge of how to scale the existing enumeration techniques. We show that the use of distributed computing techniques might offer practical solutions to previously unsolvable problems and opens new perspectives for the automated analysis of software product lines.

1. INTRODUCTION

Modeling and reasoning about features and their constraints is a crucial activity in *software product line* (SPL) engineering [2]. Products of an SPL are derived from a combination of features (aka *configuration*) through the assembly of corresponding and reusable artefacts. In an SPL context, the formalism of *feature models* is widely used [2, 4]. Feature models encode the variability of an SPL in terms of mandatory, optional and exclusive features as well as propositional constraints over the features. Feature models delimit the scope of a configurable system (i.e., an SPL) and formally document what *configurations* are supported. Once specified, feature models can be used for model checking an SPL [24], for testing SPLs, for automating product configuration [8], or for computing relevant information [4].

The large number of configurations that a feature model can encode makes the manual analysis an error prone and costly task. Computer-aided mechanisms appeared as a solution to guide and help practitioners in different software engineering tasks such as debugging, configuration or testing. This process, known as *automated analysis of feature models* [4], is a key concept when reasoning about large and complex variability-intensive systems.

In this paper we propose the use of enumeration to ease off the time required to perform repetitive and computationally hard reasoning operations over a feature model. That is, we address the problem of enumerating all configurations of a feature model so that reasoning operations can be efficiently performed afterwards. Our motivation is three-fold. First, it is not acceptable for interactive systems (e.g., Web configurators) to provide a response superior to 10 seconds [5,22] to customers. It is neither acceptable for critical configurable or re-configurable systems (e.g., dynamic SPLs) in which the response time is a critical aspect. In such contexts, reasoning mechanisms that operate over the configuration set *must* provide a fast response (e.g., see [9]). Second, some reasoning operations over feature models may be inefficient (e.g., existential quantification for slicing feature models [1], counting of the number of products) with traditional solving techniques. The use of a pre-compiled, enumerated set of configuration makes the promise of realizing operations in polytime [6]. Third, computing all configurations has an interest *per se* since it is then possible to derive all corresponding products for testing, model-checking, debugging, or measuring performance of them *individually*.

Unfortunately the enumeration of all configurations has a significant cost in space and time, precluding their practical usage. However, nowadays there exist the cloud computing wich enables the access to huge amounts of computing facilities. Concretely, we present an innovative solution based on distributed computing and big data techniques (Hadoop) that benefits from this availability of computing power.

2. PROBLEM STATEMENT

The automated analysis of feature models [4] is usually as follows. First, the model is translated into logics and reasoners are implemented using BDDs, SAT solvers, etc. Then users can perform multiple queries with reasoners such as determining the number of products or detecting dead features (i.e., features that are never present in a valid configuration). In general we expect that automated analyses for software product lines are efficient (e.g., the analysis is done in less than a second). Yet the theory states that most of the reasoning operations are computationally hard (e.g., NP-complete). The basic reason is that automated methods resolve satisfiability (SAT) problems. Hence the practical challenge is to provide efficient techniques for reasoning about the configuration set.

The more direct path is to enumerate all (valid) configurations of a feature model. The idea is to have an explicit configuration set in which reasoning operations can be efficiently performed. For instance, counting the number of valid configurations then boils down to count the size of the enumerated set – instead of producing all valid assignments of a satisfiability problem. The promise is thus to improve or even guarantee response times for some reasoning operations heavily employed for configuring, testing, or model-checking product lines.

Knowledge compilation. In fact the problem of choosing the right representation of a feature model for an efficient reasoning can be seen as a *knowledge compilation problem.* Knowledge compilation is a family of approaches for addressing the intractability of a number of reasoning problems. A propositional model (and a feature model) is compiled in an off-line phase in order to support some queries in polytime. Many ways of compiling a propositional model exist such as *conjunctive (resp. disjunctive) normal form* a.k.a. CNF (resp. DNF), and *binary decision diagrams (BDDs)* which are also widely used in the field [20]. An enumerated representation of the configuration set can be seen as an alternative to CNF, DNF, and BDD.

A key aspect of the problem is that different compiled representations (e.g., CNF, DNF, BDD, enumerated) have different properties [6]. Two properties are of interest. First, the queries are supported in polytime. For example, the consistency check can be done in polytime using ordered BDD or DNF while it is not the case in CNF. Consistency check is at the heart of numerous operations for feature models, hence not having polytime operations may be a problem. Second, some transformations of the representations can be performed in polytime. For example, the negation cannot be done in polytime with CNF while it is possible with ordered BDD. The negation of a CNF formula is relevant for some reasoning operations of feature models (e.g., see [25]) and imposes some heuristics to cope with the problem.

Given a feature model, practitioners can use BDDs or SAT solver [19, 21] for then realizing some reasoning operations. As an alternate approach we propose to use an enumeration with the idea of gaining speed when reasoning.

Towards enumerating all configurations. Compiling an enumerated set of configurations of a feature model is an alternate and interesting approach for efficiently reasoning about a feature model. To the best of our knowledge there is no prior work addressing the problem (see also the related work section). It has three practical interests:

- gathering all configurations has an interest *per se.* For example it is then possible to derive all corresponding products (e.g., programs) for testing, model-checking, debugging, or measuring performance of them *individually.* A configuration set can also be used as a benchmark for approaches seeking to synthesize feature models from configurations [3, 11, 17, 23]. As a sound and complete representation of the configurations, the set can act as a ground truth for testing some automated operations [4], etc.;

- an enumerated configuration set allows some reasoning operations to be realized in polytime [6]. It can improve or guarantee the time response of automated analyses (e.g., in critical re-configurable systems [9]);

- from the configuration set, other representation (e.g., BDDs) can be compiled for efficient queries (e.g., in polytime). Compilations (or transformations) themselves of other representations can be realized in polytime [6].

Two important issues remain before an actual adoption in practice. First, this operation is very costly in time and space. We address the challenge with a distributed solution for pre-compiling a comprehensive, enumerated configuration set of a feature model. Second, it is unclear what could be the speed up benefits when reasoning with an enumerated set. We perform preliminary experiments on real-world feature models and show that three reasoning operations can benefit to our approach (see next section).

3. PAVIA: USING HADOOP TO GENERATE A KNOWLEDGE-BASE FOR INTERACTIVE CONFIGURATION

MapReduce is a paradigm for the analysis of the so-called big-data in a scalable manner. Algorithmically, MapReduce is built on top of the divide and conquer concept, i.e., breaking big-data into smaller chunks and processing them in parallel to obtain solutions in a distributed environment. Hadoop[1] is an open-source system that implements the MapReduce programming model for distributed computing.

Deriving all configurations (satisfying the constraints of a feature model) can be seen as the problem of generating all permutations taken by n at time where n goes from 1 to k being k the number of features of the feature model. Figure 1 shows the main steps followed by PAVIA (Paralel AnAlysis of Variability Intensive systems). A feature model representing the system is used as input. First, it is translated into a logic paradigm such as SAT for its later exploitation. Each mapper will load the SAT problem description for the verification of each configuration validity. Second, a workflow of Hadoop jobs is executed to generate the set of feature combinations in an intelligent manner, this is, some permutations are discarded to avoid futile maps operations. Finally, this set of configurations is used to perform feature model analysis such as the detection of dead features.

3.1 Preparing the reasoning engine for the configuration validity verification

A key problem when distributing the computation using Hadoop is the communication within the different components –mappers, reducers, and task managers– existing in the cloud environment where we execute it. PAVIA describes the constraints existing in a feature model by using a SAT description. This enables to store in memory the minimal set of variables required to perform the validity checks, thus being compatible with the data isolated execution model proposed by Hadoop in which, mappers are oblivious to other mappers. This is, each mapper will load the problem in memory and process a random chunk of data to analyze.

3.2 Deriving the set of valid configurations using a workflow of Hadoop jobs

[1]http://hadoop.apache.org/

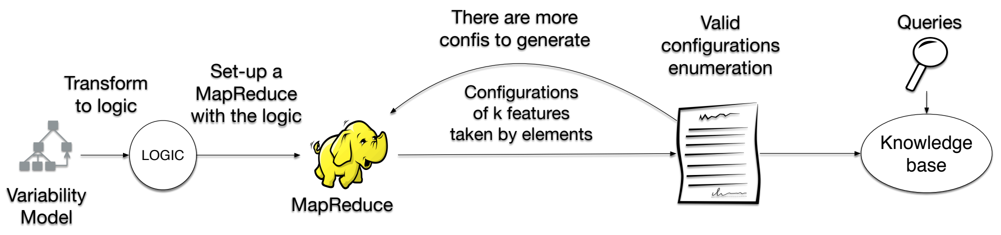
There are more confis to generate
Valid configurations enumeration
Queries
Transform to logic
Set-up a MapReduce with the logic
Configurations of k features taken by elements
LOGIC
Knowledge base
Variability Model
MapReduce

Figure 1: The PAVIA approach

PAVIA splits the problem of deriving all valid configurations encoded in a feature model in k subproblems being k the number of features. This is, in each step, we generate all combinations of k features, taken m features at a time. Thus, deriving the configurations (feature combinations) depicted by m features ($m < k$). Moreover, only valid configurations (partial or complete) are passed to the next step. This last verification is done by relying on a logic paradigm such as SAT, which prevents us to derive non-valid configurations (and to keep using them for generating more non-valid configurations). The variables used by each Hadoop mapper are defined by the tuple:

$$PAVIA =< F, Conf_{j-1}, FC, C >$$

where:

- F is a set of variables, $f_i \in F$, representing the set of features existing in the variability intensive system feature model. Also, the value of i is used as key for a feature across the subproblems.
- $Conf_{j-1}$ is the set of partial configurations coming from the previous step. Note that each mapper will only take a split of the input data.
- FC is the set of constraints that define the different relationships between different features (e.g. if the i_{th} feature is a mandatory child feature of the j_{th} feature, then $f_i <=> f_j$) according to the mapping presented in [4].
- C is the set of constrains imposed by the input configuration $Conf_{j-1}$. This is, if c_i (the feature f_i in the configuration) is present, a constraint is introduced in the problem forcing $f_i = 1$.

Inside each mapper, we combine the set of previously generated partial configurations resulting from the previous iteration ($Conf_{j-1}$) with the set of features from the model. Later, we check the validity of those newly generated configurations and send the valid ones to the reducers which will save them for the next step. Also, in the case of not being a valid partial configuration a resulting configuration, it is discarded. The Hadoop workflow ends when all new configurations are non-valid. Note that in this solution, a reducer only groups partial key-value pairs, i.e, a reducer does not compute on grouped key-value pairs.

Figure 2 depicts an example of the process that PAVIA applies to generate the set of valid partial configurations (arrows) as well as the set of complete configurations (marks) from a feature model. In our example, we did not include the root feature for the sake of simplicity. In the first iteration, we generate the set of valid combinations taken one at time which is an enumeration of the model features. This is, A; B and; C, detecting that the feature B is valid as both partial and complete configuration thus, we save it in the set of final products. All the three features are valid as partial configurations being used as input in the 2nd iteration. This is, only valid partial configurations are passed through to the next iteration while valid complete configurations are saved as result of the process.

Later in the 2nd iteration we combine the valid configurations from the first iteration with all the features which have a greater value than the last added one (this is done by using Hadoop keys.) Therefore, we test AB and AC for the A feature; and BC for the second. In this step, we also note that we cannot combine C with any other feature with a greater value so we discard it. This process is repeated until we detect that the feature combination ABC cannot be extended.

3.3 Preliminary results

We performed preliminary experimentations to evaluate the potential benefits of a distributed, enumerative-based solution. We considered 182 feature models from the SPLOT repository.

First, we generated the set of valid configurations using PAVIA in a single cluster machine of 20 cores *versus* a single-threaded solution such as the provided by the FaMa tool [26]. For feature models with small number of configuration, PAVIA induces a slight overhead and is inferior to a non-distributed solution. The single threaded solution starts experiencing scalability problems for large configurations. Here PAVIA balances the computation between multiple machines enabling the execution within the time-limit (30 minutes in our settings). Overall PAVIA was able to cope with a 32.4% (59 models) more than the FaMa implementation for the 30 minutes timeout.

As a second experiment, we compared the time required to execute some feature model operations versus the time of exploiting the database generated by Hadoop. Specifically, we measured the time of executing the operation using FaMa relying on the JavaBDD and Sat4j solvers. For exploiting the database we used simple Bash scripts for iterating over the configuration files (e.g. *"wc -l"* for counting the number of configurations). We observed that exploiting the enumeration is:

- 8.605 times faster than JavaBDD and 4.333 times faster than Sat4j for the core features operation.
- 13.880 times faster than JavaBDD and 125182.992 times faster than Sat4j for the dead features operation.
- 13.769 times faster using JavaBDD and 77060.12 times faster using Sat4j for the products enumeration question.

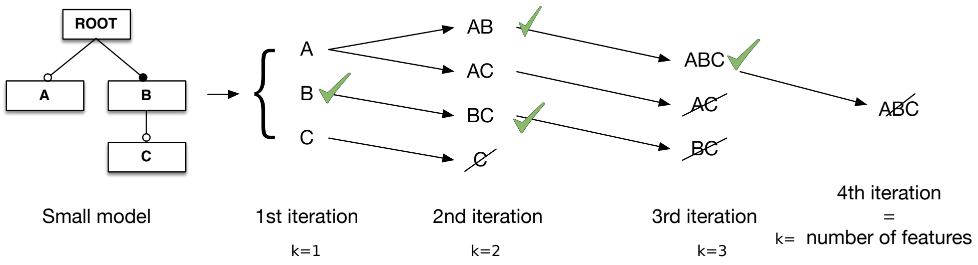

Figure 2: Process followed by PAVIA to derive all valid partial and full configurations (each iteration is a Hadoop job).

While results show a potential improvement, a more exhaustive experimentation is still required to determine when the overhead introduced by Hadoop pays off.

4. RELATED WORK

Parallel analysis of feature models. There are other researchers coping with the problematic of scaling over when analyzing feature models. Lopez-Herrejon et al. [16] compared a PPGS, Parallel Prioritized product line Genetic Solver, algorithm for prioritized software testing with the greedy algorithm pICPL, prioritized-ICPL, and shows statical analysis and results on feature models from different sources - SPL conqueror [13], and SPLOT website, to conclude while PPGS permits obtaining shorter covering arrays, pICPL is faster. In our research we tackle a different problem but also having in mind that distributing analysis tasks helps in terms of time to perform the analysis; even though, since PAVIA is a MapReduce application on feature models, we step out the distributed and parallel computing issues. Guo et al. [8] addressed the problem of multi-objective combinatorial optimization. They developed parallelization algorithms and show substantial gains in three case studies.

The use of SAT solvers or BDD has been widely considered and provide good practical results [15,19,21]. Our preliminary experiments suggest that an enumerative strategy can improve more traditional techniques. As future work we plan to investigate for which reasoning operations (e.g., T-wise configurations) an enumerative approach brings benefits (if any).

Divide and conquer. There have been proposals to introduce divider and conquer strategies when coping with feature models. Basically this effort has been focusing on determining diagnoses and redundancies; FastDiag and FM-Core. FastDiag [7] is a divide-and-conquer algorithm that supports the efficient determination of minimal diagnoses without the need of having conflict sets available. FM-Core is an algorithm which focuses on the determination of *minimal cores*, i.e., redundancy-free subsets of a constraint set. In this paper we propose to go further in the pre-computation of relevant structures for reasoning.

Enumeration of solutions. The problem of enumerating all configurations of a FM can be seen as an instance of *All-SAT* or *model enumeration* [12,14,18,27]. The problem is considered as important in the SAT community, with numerous applications (e.g., computation of backbones, unbounded model checking, knowledge compilation). A first contribution of this paper is to show that the problem is also worth studying in the context of software product lines. A second contribution is to investigate the potential of distributed computations (here Hadoop) in practical settings for scaling up.

5. CONCLUSION AND FUTURE WORK

In this paper we considered the problem of exploiting the result of enumerating all configurations of a feature model. We motivated the practical importance of the problem and identified connection with the knowledge compilation problem [6]. An enumeration can be seen as an efficient representation of feature model configurations supporting polytime reasoning operations (queries and transformations). We then addressed the problem of compiling in an off-line phase the enumerated set.

Perspectives. The use of distributed, big data techniques looks promising. It offers scalable solutions to previously unpractical problems. This opens new perspectives for the automated analysis of feature models since polytimes queries might be now be considered.

An interesting direction is to determine when the precompilation (in an offline phase) pays off, that is, for which automated analyses and hardnesses of feature models our approach is worth using. There might be also cases in which our approach has drawbacks and no improvement can be observed. More generally an open research question can be formulated: *given a reasoning operation and a feature model, is an enumeration-based approach more efficient than the traditional use of solvers?* This question is particularly relevant in scenarios involving costly and repetitive reasoning operations – for instance, the re-configuration of a dynamic software product line [10] based on several objectives.

We are also aware an enumeration-based approach is simply not possible for variability models with very large configuration space. A general perspective is thus to understand the practical limits of an enumeration, i.e., for which classes of feature models we are unable to enumerate using large amounts of computing capabilities. Furthermore it should be noted that our approach can be of interest even for rather small feature models since the underlying reasoning operations are simply very costly (e.g., multi-objective optimization in time pressure settings). Future work will involve identifying the boundaries and barriers of an enumeration-based approach. In cases a comprehensive enumeration is simply not possible, an interesting direction is to partially enumerate configurations.

We also believe the work is of interest for the SAT community. The problem can be seen as an instance of *All-SAT* or *model enumeration* [12,14,18,27]. In the context of software product lines the problem exhibits specific properties (in terms of motivation and hardnesses of the instances) and is worth studying.

Acknowledgements

This work has been partially supported by the Andalussian goverment under the COPAS project (P12-TIC-1867) and the Spanish goverment under the BELi project (TIN2015-70560-R).

6. REFERENCES

[1] M. Acher, P. Collet, P. Lahire, and R. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP)*, 78(6):657–681, 2013.

[2] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer-Verlag, 2013.

[3] G. Bécan, M. Acher, B. Baudry, and S. Ben Nasr. Breathing ontological knowledge into feature model synthesis: an empirical study. *Empirical Software Engineering*, 2015.

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35(6), 2010.

[5] R. Bryant, R. Hoffman, and S. Kahn. World wide web end user response time monitor, June 20 2000. US Patent 6,078,956.

[6] A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.

[7] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 26(1):53–62, 2012.

[8] J. Guo, E. Zulkoski, R. Olaechea, D. Rayside, K. Czarnecki, S. Apel, and J. M. Atlee. Scaling exact multi-objective combinatorial optimization by parallelization. In *ASE '14*, pages 409–420, 2014.

[9] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In *In PETO CONFERENCE, DTU-TRYK*, pages 131–138, 2004.

[10] S. O. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *IEEE Computer*, 41(4):93–95, 2008.

[11] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. On extracting feature models from sets of valid feature combinations. In *FASE'13*, 2013.

[12] S. Jabbour, J. Lonlac, L. Sais, and Y. Salhi. Extending modern SAT solvers for models enumeration. In *15th IEEE International Conference on Information Reuse and Integration*, pages 803–810, 2014.

[13] M. F. Johansen, Ø. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *16th International Software Product Line Conference, SPLC '12*, pages 46–55, 2012.

[14] S. Khurshid, D. Marinov, I. Shlyakhter, and D. Jackson. A case for efficient solution enumeration. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 272–286. Springer Berlin Heidelberg, 2004.

[15] J. H. J. Liang, V. Ganesh, K. Czarnecki, and V. Raman. Sat-based analysis of large real-world feature models is easy. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015*, pages 91–100, 2015.

[16] R. E. Lopez-Herrejon, J. Javier Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 1255–1262, 2014.

[17] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, and A. Egyed. An assessment of search-based techniques for reverse engineering feature models. *Journal of Systems and Software*, 2014.

[18] J. Marques-Silva, M. Janota, and I. Lynce. On computing backbones of propositional theories. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 15–20. IOS Press, 2010.

[19] M. Mendonca, A. Wasowski, and K. Czarnecki. SAT-based analysis of feature models is easy. In *SPLC'09*, pages 231–240. IEEE, 2009.

[20] M. Mendonca, A. Wasowski, K. Czarnecki, and D. Cowan. Efficient compilation techniques for large scale feature models. In *GPCE'08*, pages 13–22. ACM, 2008.

[21] R. Pohl, V. Stricker, and K. Pohl. Measuring the structural complexity of feature models. In *ASE'13*, 2013.

[22] U. Raz, Y. Volk, and S. Melamed. Method and system for decreasing the user-perceived system response time in web-based systems, Nov. 1 2001. US Patent App. 09/747,260.

[23] S. She, U. Ryssel, N. Andersen, A. Wasowski, and K. Czarnecki. Efficient synthesis of feature models. *Information and Software Technology*, 56(9), 2014.

[24] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys*, 2014.

[25] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 254–264. IEEE, 2009.

[26] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. Fama framework. In *12th Software Product Lines Conference (SPLC)*, 2008.

[27] Y. Yu, P. Subramanyan, N. Tsiskaridze, and S. Malik. All-sat using minimal blocking clauses. In *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pages 86–91, Jan 2014.