Automated completion of partial configurations as a diagnosis task
Using FastDiag to improve performance

Cristian Vidal-Silva[1], José Ángel Galindo[2], Jesús Giráldez-Cru[3], and David Benavides[2]

[1] Departamento de Administración, Facultad de Economía y Administración, Universidad Católica del Norte, Antofagasta, Chile
cristian.vidal@ucn.cl
[2] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Sevilla, España
{benavides, jagalindo}@us.es
[3] Andalusian Research Institute DaSCI "Data Science and Computational Intelligence", Universidad de Granada, Granada, España
jgiraldez@ugr.es

Abstract. The completion of partial configurations might represent an expensive computational task. Existing solutions, such as those which use modern constraint satisfaction solvers, perform a complete search, making them unsuitable on large-scale configurations. In this work, we propose an approach to define the completion of a partial configuration like a diagnosis task to solve it by applying the FastDiag algorithm, an efficient solution for preferred minimal diagnosis (updates) in the analyzed partial configuration. We evaluate our proposed method in the completion of partial configurations of random medium and large-size features models and the completion of partial configurations of a feature model of an adapted version of the Ubuntu Xenial OS. Our experimental analysis shows remarkable improvements in our solution regarding the use of classical CSP-based approaches for the same tasks.

Keywords: Partial configuration · completion · FastDiag.

1 Introduction

Configuration technology is a successful application of artificial intelligence (AI) in the industry [9]. By applying configuration technology, the development and maintenance costs of critical functionalities (features) notably reduce enabling the mass customization realization [11]. Software product line (SPL) is an application domain for the mass-customization of software products [8].

Software product line engineering (SPLE) promotes the mass customization of software products (configurations) by identifying common and reusable features (e.g., functionalities) for the satisfaction of individual consumer requirements, and taking advantage of using a defined production framework [1, 6].

SPLE relies on efficient mechanisms to detect and diagnose anomalies in configurations, i.e., finding configurations that violate some constraints of the SPL and explaining the reasons for such inconsistency. To this purpose, feature models (FMs) have been proposed as a compact abstraction of families of products since they allow to represent all the existing features in the family and constraints among them [17]. FMs represent the valid product configurations of a software product family [6]. Although in this work we consider basic FMs [4], our proposed solution can be directly applied to more complex FMs, such as cardinality-based FMs [14] and attributed FMs [18], as well as to other configurations approaches supported by some reasoning technology. We use FMs as an example to illustrate our approach.

The completion of partial configurations consist of finding the set of non-selected components necessary for getting a complete configuration. In FM configurations, each feature is decided to be either present or absent in the resulting products, whereas in partial configurations, some features are undecided. The completion of partial configuration is a non-trivial and computationally expensive task due to the existence of constraints among features of FMs [15], and more expensive in large-scale FMs. Configurations can result in misconfigurations (i.e., non-valid configurations) which can impact on the system availability [25]. Unavailability of the Facebook platform [7], service-level problems of Google [3], and invalid operation of Hadoop clusters [19] are known misconfiguration examples.

In the literature there exist efficient algorithms for the automated analysis and diagnosis of FMs, such as FastDiag and FlexDiag [8, 11, 12]. FastDiag and FlexDiag rely on encoding the FM constraints into the formal representation of reasoning technology for diagnosis in those configurations using off-the-shelf solvers (e.g. Constraint Satisfaction Problem and SATisfiability Problems, that is, CSP and SAT, respectively). Existing computer-assisted methods for the completion of partial configurations, such as modern CSP solvers, often apply computationally expensive complete search functions [24]. Hence, to find a consistent FM configuration of FMs with n features requires exploring 2^n possible configurations in the worst case. Moreover, CSP and SAT solver solutions can be minimal, but they not always represent the preferred configuration [22].

In this work, we define the completion of partial configurations as diagnosis tasks to solve them by using an efficient diagnosis algorithm. Specifically, we use the diagnosis algorithm FastDiag to evaluate its performance regarding applying a traditional CSP-based approach. Our experiment consists of random products of a set of FMs, both randomly generated by the Betty toolkit, and partial configurations of the FM of a Ubuntu Xenial version. The obtained results show that our proposal is several orders of magnitude faster than the applied traditional CSP-based approach. Thus, Our contributions are the following:

- We define the completion of partial configurations as a diagnosis task. This approach allows us to directly apply FastDiag to get a consistent configuration from a predefined set of features (i.e., from a partial configuration).
- We provide a public available implementation of our solution in the FaMa platform [5], as well as a set of available models and configurations.

The rest of the article is organized as follows. Section 2 describes preliminary background on FM and existing diagnosis solutions. Section 3 establishes the background to define the completion of partial configurations as diagnosis problems. Section 4 defines case studies and presents the application results of our solution. Some related works are described in Section 5. Finally, Section 6 concludes and proposes future work.

2 Preliminaries

2.1 Feature models and completion of partial configurations

An FM is a tree-like hierarchical representation of features and their constraints for a family of products. The following types of constraints exist in basic FMs: (a) parent-children or inclusion relationships, and (b) cross-tree constraints (CTC). Four kinds of inclusion relationships exist: (i) *mandatory* (the parent requires its child, and *vice versa*), (ii) *optional* (the parent does not require its child), (iii) *inclusive*-OR (the parent requires at least one of the set of children), and (iv) *alternative*-XOR (the parent requires exactly one of the set of children). CTC of traditional FMs are (v) *requires* (a feature requires another), and (vi) *excludes* (two features cannot be in the same configuration). Figure 1 illustrates a FM with a root feature *Debian* that has the mandatory children *texteditor*, *bash* and *gui*. Feature *texteditor* has an inclusive set of children features *vi*, *gedit* and *openoffice.org-1*, and *openoffice.org-1* also has two optional children features *openoffice.org-1.1* and *openoffice.org-1.2*. Feature *gui* has an inclusive set of children features *gnome* and *kde*. Feature *gnome* is required by feature *openoffice.org-1*.

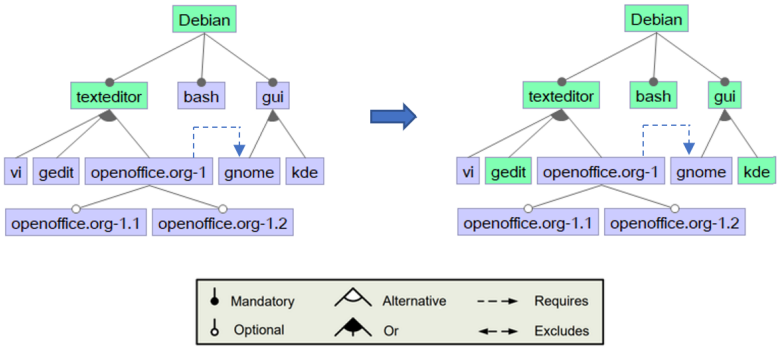

Fig. 1. An example of a partial configuration completion in the Debian FM.

Figure 1 illustrates this problem (the features in green represent selected features). The partial configuration in the left $\{Debian, texteditor\}$ is extended to the complete configuration $\{Debian, texteditor, gedit, bash, gui, kde\}$ in the right. We construct these models using FeatureIDE [20].

2.2 FM Configuration and Diagnosis Tasks

An FM configuration task (F, D, C) consists of setting the values of a set of features $F = \{f_1, \ldots, f_n\}$ in a common domain D to satisfy a set of configuration constraints $C = CF \cup CR$. CF represents the FM base knowledge (i.e., constraints among the features) and CR the user preferences (i.e., desired features in the product) [8], and D is {`true`, `false`} usually. Hence, a complete configuration represents a setting of each feature f_i in F respecting the configuration constraints of C.

Configurations that violate the FM constraints require diagnosis operations for identifying solutions. For a consistent knowledge base AC, and a non-consistent configuration S, a diagnosis task (S, AC) gives a set of constraints or diagnosis $\Delta \subseteq S$ such that $(AC - \Delta)$ is consistent. Hence, S is the set of selected features in a configuration. The presence or absence of a set of features $S = \{f_1, \ldots, f_k\}$ can be expressed as the constraint $\forall i\ 1 \leq i \leq k$, $f_i =$ `true` or `false`, respectively. Δ is minimal if $\neg\exists \Delta' \subset \Delta$ satisfying the diagnosis property in AC.

3 Minimal completion of configurations by diagnosis

As was mentioned in the previous section, to proceed with an FM diagnosis, FastDiag receives the parameters S and AC, that is, the user preferences and the FM knowledge base that contains S. The completion of a partial configuration is a diagnosis task to find the preferred minimal set of features to select for getting a full configuration. Hence, the main task to apply FastDiag for diagnosis a preferred minimal completion is to define the knowledge base and the suspicious set of constraints in conflict.

An FM formally represents a set of features F and a set of constraints C. A partial configuration can be seen as a set of assigned features S, i.e., $S \subset F$. Likewise, we define the set of unassigned features nS as $nS = (F - S) \neq \emptyset$. The partial configuration S is valid if $C \cup S$ is consistent, which, for the sake of clarity, we always assume to hold. To find the remaining features for a complete configuration, we run FastDiag with $S = nS$, and a knowledge base $C \cup S$. We assign Boolean values to the components of S and nS for consistency checks in the FM. FastDiag returns a preferred minimal set $\Delta \subseteq nS$ of features necessary for the completion of the partial configuration S. In summary, we define a FastDiag application for diagnosis features for the completion of partial products. Table 1 gives our definition for that task. The sets S and nS represent the selected and non-selected features in the partial configuration p respectively, and C is the set of base constraints in the FM. Because FastDiag works on constraints in a reasoning solver tool such as CSP and SAT, our solution is not restricted to work only for the FMs completion. We suggest to read [10, 13] for more details of FastDiag.

FastDiag gives an ordered by preference set of features to update using a lexicographical order by default. Our solution can use personalized options for selecting features such as randomly, the nearest feature to some already chosen feature, or based on a priority ranking regarding previous configurations.

Table 1. Diagnosis-based solution for the completion of a partial configuration using FastDiag.

Analysis operation	Property check	Explanation (Diagnosis)
Completion of Partial Configurations	Diagnosis in nS (set of non-selected features)	$FastDiag(nS, C \cup S \cup nS)$

4 Empirical Evaluation

To evaluate the performance of our solution, first, we generate a set of random FMs using the Betty tool-suite [23] to define the number of features and structure of randomly generated FMs. Betty also allows us to define the number of CTC in the model. In our experiment, we generate models with the following number of features $|F| = \{50, 100, 500, 1000, 2000, 5000\}$ and the following percentages of CTC $c = \{5, 10, 30, 50, 100\}$. For each model, we generate partial configurations with the following percentages of assigned features $a = \{10, 30, 50, 100\}$. We generate 10 random instances for each model and partial configuration.

Table 2. Avg. time (in milliseconds) on the completion of partial configuration of randomly generated Betty FMs by the number of features n.

n	CSP-based app	FastDiag app	% speed-up
50	98.00	**97.90**	0.10
100	109.06	**104.63**	4.06
500	200.09	**171.60**	14.24
1,000	405.89	**258.26**	36.37
2,000	1,392.27	**411.01**	70.48
5,000	15,677.93	**808.61**	94.84
All	2,980.54	**308.67**	36,58

Our proposal is evaluated using FastDiag in the FaMa tool suite with the Choco CSP solver [5] for consistency checks. The CSP-based approach uses the same solver. In what follows, we report the results comparison of both approaches. In each results table, the best values are marked in bold.

In Table 2 we report the average solving time of the CSP-based and the FastDiag approach, aggregating the results by the number of features in the random models. In Tables 3 and 4, we report the same results aggregated by the percentage of CTC and the percentage of features in the partial configuration, respectively. The last column ($\%speed-up$) of each table shows the percentage of improvement of our solution regarding the CSP-based approach. The last row presents the average results.

The first observation is that in all the comparisons, the FastDiag diagnosis solution is faster than the CSP-based approach. In general, there are noticeable differences, in some cases with a speed-up greater than 19x (see $n = 5000$ in Table 2). Hence, the performance improvements are bigger as the number of features in the FM increase. This is possibly due to the complete search of the CSP solver that scales exponentially. In contrast, our solution seems to scale much better. Notice that the speed-ups in Table 2 increases for greater values

of n. On the contrary, as Tables 3 and 4 show, the number of CTC and size of the partial configuration seem to have a low effect on the performance of both solutions. However, there are two remarkable effects. First, the speed-up of FastDiag slightly decreases as the number of CTC increases. This suggests that if the number of constraints is exponentially big, there may be cases in which both approaches perform similarly. Second, the speed-up of FastDiag slightly decrease as the partial configuration becomes smaller. This suggests that the larger the size of such a partial configuration, the bigger the differences between FastDiag and the CSP-based approach.

Table 3. Avg. time (in milliseconds) on the completion of partial configuration of randomly generated Betty FMs by the % of CTC *cc*.

c	CSP-based app	FastDiag app	% speed-up
5	2,953.98	**275.80**	90,67
10	2,993.21	**282.57**	90,56
30	2,971.61	**303.87**	89,77
50	2,937.65	**308.72**	89,49
100	3,046.24	**372.38**	87,78
All	2,980.54	**308.67**	89,64

Table 4. Avg, time (in milliseconds) on the completion of partial configuration of randomly generated Betty FMs by the % of features in the partial configuration a.

a	CSP-based app	FastDiag app	% speed-up
10	2,954.96	**318.23**	89.23
30	2,973.43	**315.03**	89,41
50	2,966.08	**306.58**	89.66
100	3,027.68	**294.82**	90,26
All	2,980.54	**308.67**	90,26

Figure 2 shows the scatter plot of both approaches, i.e., the solving time of the CSP-based approach in the X axis versus the FastDiag approach in the Y axis. This plot confirms the expected performance from the aggregated results in the previous tables. In particular, we can observe that the solution based on diagnosis algorithm scales quite well with the number of features of the generated model, whereas there are only small differences when this solution is compared with respect to other parameters, such as the number of CTC or the size of the partial configuration.

As a second performance evaluation, we generated an FM and partial products for the Ubuntu Xenial OS. We generate five valid partial products of 5%, 10%, and 15% of a complete and valid configuration for that FM, respectively. Then, we apply the CSP-based approach and FastDiag for the completion of those products. Table 5 and Figure 3 show the computation results. Like in the previous report, speed-up percentages exist in these tests that confirm the efficiency of our solution.

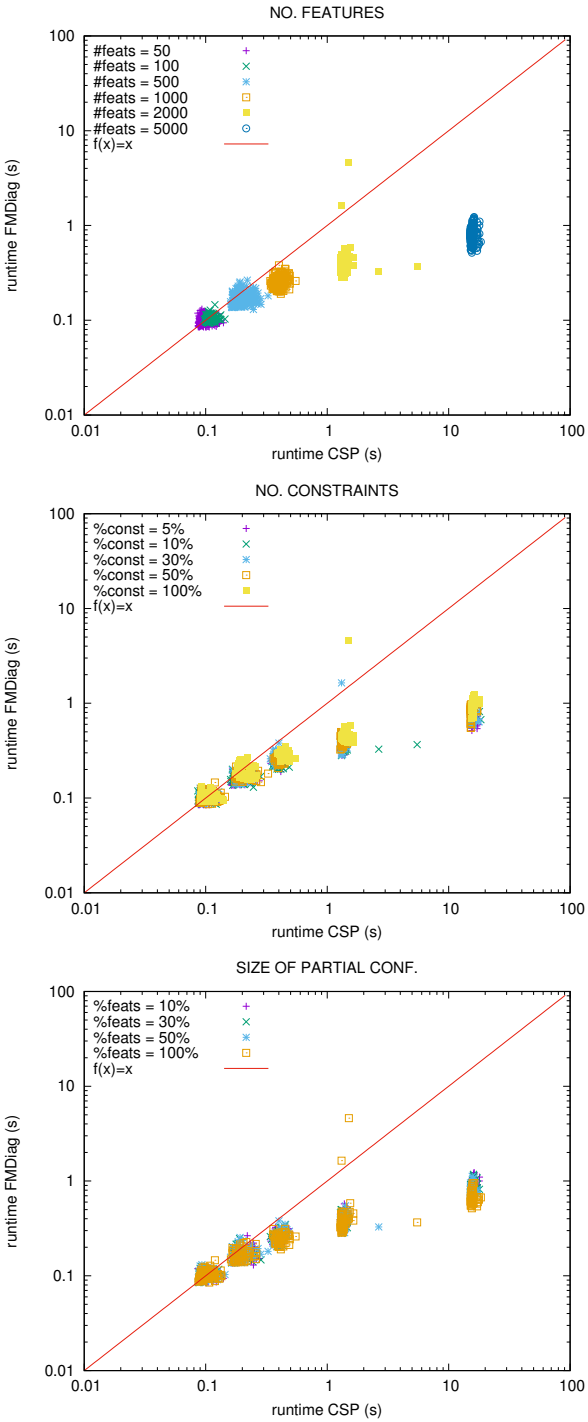

Fig. 2. Scatter plot of CSP-based versus FastDiag approach on the completion of partial configuration of randomly generated Betty FMs (in seconds), aggregated by the number of features (top left), by the percentage of CTC (top right), and by the percentage of features in the partial configuration (bottom).

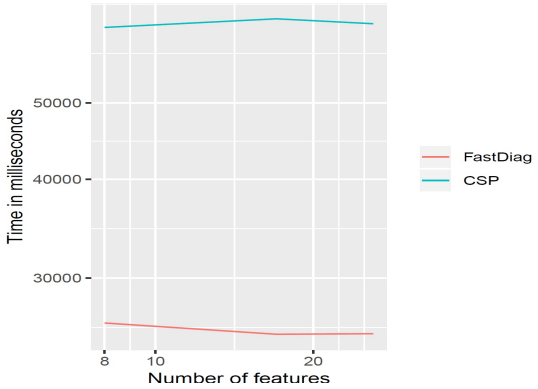

Fig. 3. Runtime execution for the completion of partial products for a FM of Ubuntu Xenial.

Table 5. Avg. solving time (in milliseconds) on the completion of partial configuration of randomly generated partial configuration of an FM for a reduced version of the Ubuntu Xenial OS by the percentage of features in the partial configuration.

$\%Features$	CSP-based app	FastDiag app	% speed-up
5	62,297.05	**26,318.14**	57,75
10	63,885.43	**25,467.03**	60,14
15	62,973.01	**25,508.02**	59,49
All	63,051.83	**25,764.40**	59,13

All experiments were executed in an Intel(R) Core (TM) i7-3537U CPU @ 2.00 GHz with 4 GB RAM using a Windows 10 64 bits operating system.

5 Related Work

Reiter [21] introduces the Hitting Set Directed Acyclic Graph for diagnosis using a breadth-first search on conflict sets. Bakker et al. [2] apply a model-based diagnosis to identify the set of relaxable constraints on conflict sets in a CSP context. Junker [16] proposes QuickXplain, a divide-and-conquer approach to significantly accelerate the conflict detection on over-constrained problems. Following the QuickXplain divide-and-conquer strategy, Felfernig et al. [10] present FastDiag for efficient diagnosis solutions on customer requirements in consistent configuration knowledge bases. The work of [8] review and apply FastDiag on the FM diagnosis, and [12, 11] describe FlexDiag as a FastDiag extension for anytime diagnosis scenarios and apply both algorithms for the FMs diagnosis. Felfernig et al. [8] also highlight an advantageous property of FastDiag regarding existing diagnosis approaches: FastDiag is a direct-diagnosis solution without a preceding conflict detection, that is, FastDiag uses a conflict-independent search strategy [9].

6 Conclusions

In this work, we defined a solution for the completion of partial configurations as a diagnosis problem that allows us to apply FastDiag to this problem. Experimental results with FMs show that this approach improve a traditional solution approach in several orders of magnitude, achieving speed-ups of more than 19x in some cases. Therefore, FastDiag also represents an efficient solution for the completion of configurations in software product lines.

As future work, we plan to adapt these ideas to FlexDiag in real-time scenarios with predefined time limits and acceptable trade-offs between the diagnosis quality and efficiency of the diagnostic reasoning. Moreover, we plan to evaluate these diagnosis algorithms with other combinatorial encoding paradigms, such as SAT or SMT.

Acknowledgements

This work has een partially funded by the EU FEDER program, the MINECO project OPHELIA (RTI2018-101204-B-C22); the TASOVA network (MCIU-AEI TIN2017-90644-REDT); and the Junta de Andalucia METAMORFOSIS project.

References

1. Apel, S., Batory, D., Kstner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer (2013)
2. Bakker, R.R., Dikker, F., Tempelman, F., Wognum, P.M.: Diagnosing and solving over-determined constraint satisfaction problems. pp. 276–281 (1993)
3. Barroso, L.A., Hoelzle, U.: The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers, 1st edn. (2009)
4. Batory, D.: Feature Models, Grammars, and Propositional Formulas. Proceedings of the 9th International Conference on Software Product Lines pp. 7–20 (2005). https://doi.org/10.1007/115548443
5. Benavides, D., Segura, S., Trinidad, P., Ruiz–Cortés, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: Proceeding of the 1st International Workshop on Variability Modelling of Software-intensive Systems (VAMOS). pp. 129–134 (2007)
6. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Journal of Information Systems **35**(6), 615–636 (2010)
7. Facebook: More details on today's outage. http://www.splot-research.org/, accessed: 2018-13-05
8. Felfernig, A., Benavides, D., Galindo, J., Reinfrank, F.: Towards anomaly explanation in feature models. In: Proceedings of the 15th International Configuration Workshop (2013)
9. Felfernig, A., Hotz, L., Bagley, C., Tiihonen, J.: Knowledge-based Configuration: From Research to Business Cases. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn. (2014)

10. Felfernig, A., Schubert, M., Zehentner, C.: An efficient diagnosis algorithm for inconsistent constraint sets. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing **26**(1), 53–62 (2012)
11. Felfernig, A., Walter, R., Galindo, J.A., Benavides, D., Erdeniz, S.P., Atas, M., Reiterer, S.: Anytime diagnosis for reconfiguration. Journal of Intelligent Information Systems (2018)
12. Felfernig, A., Walter, R., Reiterer, S.: Flexdiag: Anytime diagnosis for reconfiguration. In: Proceedings of the 17th International Configuration Workshop (2015)
13. Fernández-Amorós, D., Heradio, R., Cerrada, J.A., Cerrada, C.: A scalable approach to exact model and commonality counting for extended feature models. IEEE Trans. Software Eng. **40**(9), 895–910 (2014). https://doi.org/10.1109/TSE.2014.2331073, https://doi.org/10.1109/TSE.2014.2331073
14. Gómez, A., Ramos, I.: Automatic tool support for cardinality-based feature modeling with model constraints for information systems development. In: Information Systems Development, Business Systems and Services: Modeling and Development [Proceedings of ISD 2010, Charles University in Prague, Czech Republic, August 25-27, 2010]. pp. 271–284 (2010). https://doi.org/10.1007/978-1-4419-9790-6_22, https://doi.org/10.1007/978-1-4419-9790-6_22
15. Ibraheem, S., Ghoul, S.: Software evolution: A features variability modeling approach. Journal of Software Engineering **11**, 12–21 (2017)
16. Junker, U.: QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In: Proceedings of the 19th National Conference on Artifical Intelligence (AAAI). pp. 167–172 (2004)
17. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University (1990)
18. Karataş, A.S., Oğuztüzün, H., Doğru, A.: From extended feature models to constraint logic programming. Science of Computer Programming **78**(12), 2295 – 2312 (2013). https://doi.org/https://doi.org/10.1016/j.scico.2012.06.004, http://www.sciencedirect.com/science/article/pii/S0167642312001153, special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of FSEN 2011)
19. Li, J.Z., He, S., Zhu, L., Xu, X., Fu, M., Bass, L., Liu, A., Tran, A.B.: Challenges to error diagnosis in hadoop ecosystems. In: Proceedings of the 27th Large Installation System Administration Conference (LISA). pp. 145–154 (2013)
20. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G.: Mastering Software Variability with FeatureIDE. Springer (2017)
21. Reiter, R.: A theory of diagnosis from first principles. AI Journal **23**(1), 57–95 (1987)
22. Riener, H., Fey, G.: Exact diagnosis using boolean satisfiability. In: Proceedings of the 35th International Conference on Computer-Aided Design. ICCAD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2966986.2967036, https://doi.org/10.1145/2966986.2967036
23. Segura, S., Galindo, J.A., Benavides, D., Parejo, J.A., Ruiz-Cortés, A.: Betty: Benchmarking and testing on the automated analysis of feature models. In: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems. pp. 63–71 (2012)

24. White, J., Benavides, D., Schmidt, D.C., Trinidad, P., Dougherty, B., Cortés, A.R.: Automated diagnosis of feature model configurations. Journal of Systems and Software **83**(7), 1094–1107 (2010)
25. Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L.N., Pasupathy, S.: An empirical study on configuration errors in commercial and open source systems. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 159–172 (2011)