How Easy is SAT-Based Analysis of a Feature Model?

Elias Kuiter
kuiter@ovgu.de
University of Magdeburg, Germany

Tobias Heß
tobias.hess@uni-ulm.de
University of Ulm, Germany

Chico Sundermann
chico.sundermann@uni-ulm.de
University of Ulm, Germany

Sebastian Krieter
sebastian.krieter@uni-ulm.de
University of Ulm, Germany

Thomas Thüm
thomas.thuem@uni-ulm.de
University of Ulm, Germany

Gunter Saake
saake@ovgu.de
University of Magdeburg, Germany

ABSTRACT

With feature-model analyses, stakeholders can improve their understanding of complex configuration spaces. Computationally, these analyses are typically reduced to solving satisfiability problems. While this has been found to perform reasonably well on many models, estimating the efficiency of a given analysis on a given model is still difficult. We argue that such estimates are necessary due to the heterogeneity of feature models. We discuss inherently influential factors and suggest potential algorithmic solutions.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; • **Theory of computation** → *Automated reasoning.*

KEYWORDS

feature modeling, SAT solving, algorithm selection

ACM Reference Format:
Elias Kuiter, Tobias Heß, Chico Sundermann, Sebastian Krieter, Thomas Thüm, and Gunter Saake. 2024. How Easy is SAT-Based Analysis of a Feature Model?. In *18th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2024), February 07–09, 2024, Bern, Switzerland*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3634713.3634733

1 INTRODUCTION

Feature models [7, 10, 21] describe the user-visible characteristics, known as *features*, of software product lines (SPLs) [6, 45]. A *configuration* of features is valid when it fulfills all feature dependencies, known as *constraints* [6]. The valid configurations of a feature model usually form large configuration spaces [21, 50], which quickly become difficult to comprehend. Thus, automated feature-model analyses have been proposed [2, 8, 9, 36, 46, 60], with which stakeholders can improve their understanding about a feature model (e.g., to spot modeling errors or guide business decisions). Furthermore, feature-model analyses enable more advanced SPL analyses, which support several activities in the software development life cycle (e.g., design [8, 52], implementation [32, 56], testing [29, 35], and economical estimates [15]).

VaMoS 2024, February 07–09, 2024, Bern, Switzerland
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0877-0/24/02.
https://doi.org/10.1145/3634713.3634733

To implement such analyses, feature models are often represented as propositional formulas [6, 7, 39, 47], which are then passed to off-the-shelf analysis tools, such as satisfiability (SAT) solvers [36, 38]. This approach is usually tractable in practice, although SAT is NP-complete. In two well-known publications, this phenomenon has been empirically investigated on large collections of feature models; concluding that "SAT-based analysis of (large real-world) feature models is easy" [31, 36].

While many studies and experiences confirm this overall sentiment, feature models are also known to be heterogeneous (e.g., in terms of origin, domain, and size) [5, 50]. Indeed, there are several large feature models (e.g., the Linux kernel, Freetz-NG, or Automotive02) that still challenge state-of-the-art analysis techniques [28, 42, 44, 50, 51, 55]. This is because not all analyses are equally tractable: For example, while a single call to a SAT solver is usually cheap to compute, some analyses are difficult or impossible to phrase in terms of a single SAT call [51]. Instead, they either require several SAT calls (e.g., reasoning about edits [57], core/dead features [12, 20], type-checking [22, 23]), specialized solvers (e.g., #SAT [50] or AllSAT [18]), or algebraic reasoning (e.g., slicing [1, 26] or differencing [2]). Thus, a more conservative interpretation of previous results might be: "*many* SAT-based analyses on *most* (large real-world) feature models are *comparably* easy".

However, this naturally begs several questions: *Which* analyses are easy on *which* feature models? What does *easy* mean for feature-model analysis? What factors influence the answers to these questions? We argue that it is time to pivot from a *class-based* point of view on feature-model complexity to an *instance-based* perspective. That is, instead of making sweeping statements about the entire class of feature models, it may be more illuminating to try to estimate the difficulty of computing a given analysis for a given feature model (potentially taking into account other influential factors). With this shift in perspective, we aim to foster a more nuanced discussion of feature-model complexity in the SPL community, which takes the heterogeneity of feature models into account. Both perspectives are cases of *feature-model meta-analysis*, which provides a general framework for talking *about* properties of feature-model analyses. In the following, we outline our idea of meta-analysis, the shift in perspective we propose, and initial suggestions for working towards instance-based meta-analysis.

2 FEATURE-MODEL META-ANALYSIS

We define *feature-model meta-analysis* as the practice of asking (and answering) questions about feature-model analyses as follows:

- First, one must ask a question about a (non-)functional property of feature-model analysis. The actual analysis results are not of interest here, instead one asks for correctness or efficiency (e.g., regarding runtime, memory usage, or energy consumption). The question fixes some factors (e.g., feature model and analysis), while leaving others blank (e.g., algorithm and solver).

- Second, one must define criteria to answer the question (either exactly or an estimate) and propose an algorithm to do so.

Here, we discuss two opposed kinds of meta-analysis: *class-* and *instance-based* meta-analyses. While this distinction is not clear-cut, it serves well for demonstrating the shift in perspective we propose.

2.1 Class-Based Meta-Analysis

Class-based meta-analyses ask questions about a whole class of feature models and/or analyses. Thus, they can illuminate the feasibility of computing certain analyses on certain models. In previous work, several questions of this kind have been asked (and answered).

"Is SAT-based analysis of feature models easy?" This is an open meta-analysis question that binds few factors and can only be answered with "yes" or "no". Mendonça et al. [36] actually pose and answer a more specific variant of this question: They focus on artificial feature models and, although acknowledging repeated SAT calls, they perform only one SAT call. They conclude that analysis is indeed "easy" because they find no phase transition.

"Is SAT-based analysis of large real-world feature models easy?" Analogously, Liang et al. study the feasibility of singular SAT calls on feature models of several open-source SPLs, which are specified using the KConfig language [31]. They give new insights as to why determining feature-model satisfiability is comparably easy—still, they do not consider complex feature-model analyses or distinguish difficulty on a per-instance basis.

2.2 Instance-Based Meta-Analysis

The above meta-analyses have likely been helpful in establishing the widespread use of SAT solvers for feature-model analyses. However, they neither acknowledge the vast gap between feature models that are computationally "simple" (e.g., the graph product line [33]) or "complex" (e.g., the Linux kernel [42, 55]), nor do they distinguish how this computational complexity may depend on the computed analysis (or other, more subtle factors). We discuss briefly how to both ask and answer instance-based meta-analysis questions.

Asking Meta-Analysis Questions To acknowledge the differences in complexity between feature models, we can ask more precise questions about analysis tasks, such as: **"How much time does analysis X need on feature model Y when using solver Z?"** or **"Which algorithm is most memory-efficient for computing X on Y?"** These questions still leave room for filling in details (e.g., system specifications), so they can only be estimated—but they will yield more useful answers for a given use case than the more sweeping statements obtained with class-based meta-analyses.

The appropriate level of parametrization depends on the use case and represents a trade-off: Binding more factors allows for more accurate estimates (improving internal validity); binding less factors allows for more general settings (improving external validity) [49]. As a starting point for posing interesting questions, we list several factors that we know or suspect to influence the correctness or efficiency of feature-model analyses:

- **Feature Model** [50]: origin [5], domain, size, and expressiveness of constraints [24, 54]
- **Propositional Encoding** [6, 7, 47]: extractor (e.g., for KConfig specifications [16, 17, 42]), non-Boolean variability [11, 40, 43], CNF transformation [28, 34], and preprocessing
- **Analysis**: class (consistency, cardinality, enumeration, or algebraic) [51], the question it answers [8, 52], the chosen algorithm [12, 20], and its implementation
- **Solver** (if needed): class (e.g., SAT [36], #SAT [50], AllSAT [18], or VSAT [59]), solver parametrization (e.g., exact or approximate, optional preprocessing steps), name/version
- **Knowledge Compilation** (if needed): class (e.g., BDD [19, 37, 55] or d-DNNF [53]), name/version
- **Prior Information** (if given): existing analysis results, revisions (incremental analysis [25]), and interfaces [48]
- **Execution Environment**: CPU, RAM, and deep variability [30]

It is one purpose of feature-model meta-analysis to study the influence of these (and other) factors and how they interact. To do so, we must find techniques to answer meta-analysis questions.

Answering Meta-Analysis Questions Ideally, we want to answer instance-based meta-analysis questions without actually computing the analysis in question, which can be costly or even infeasible. Instead, one usually tries to investigate surrogate *metrics* (e.g., on an ordinal or interval scale) to estimate analysis complexity. For example, we can characterize feature models using metrics:

- **Syntactic Metrics** [50]: number of features, variables, constraints, clauses, literals; constraint size, density [31]
- **Semantic Metrics** [3, 4]: phase transition [36], community structure [41], self-similarity

While syntactic metrics are easy to compute, they seem to allow for rough estimates at most [19, 50]. Semantic metrics are probably better indicators for inherent complexity of a feature model, but are themselves usually NP-hard and could therefore be approximated.

Once we have determined suitable metrics for studying a meta-analysis question, we must also choose an algorithm to answer it. To this end, previous work uses simple criteria (e.g., "yes/no" for a phase transition [36]) or otherwise handcrafted models and hypotheses (e.g., the number of features correlates with analysis time [50]). Alternatively, machine learning techniques might be applicable, but we are not aware of any studies in this direction.

3 CONCLUSION

By pivoting from class- to instance-based meta-analysis, many directions for discussions and future work open up: What meta-analysis questions are worth asking, what factors are relevant, and how do they interact? Is feature-model complexity intrinsic, regardless of the chosen analysis or solving technique? When do knowledge compilation and incremental analysis pay off?

By improving our ability to answer instance-based meta-analysis questions, we lay a foundation for implementing *meta-analyzers* that (semi-)automatically choose the best (e.g., fastest) *analysis plan* (i.e., algorithm, solver, ...) for a given analysis task, analogous to what portfolio solvers do for SAT [58]. Thus, analysis plans render analyses into first-class objects, which we can precisely describe, manipulate, and optimize; as has been done for databases [14] and, to some degree, also been proposed for SPL analyses [13, 27].

REFERENCES

[1] M. Acher, P. Collet, P. Lahire, and R. B. France. 2011. Slicing Feature Models. In *ASE*. IEEE, 424–427.

[2] M. Acher, P. Heymans, P. Collet, C. Quinton, P. Lahire, and P. Merle. 2012. Feature Model Differences. In *CAiSE*. Springer, 629–645.

[3] R. Adamy, E. Kuiter, and G. Saake. 2023. *Exploiting Structure: A Survey and Analysis of Structures and Hardness Measures for Propositional Formulas.* Technical Report. University of Magdeburg. https://doi.org/10.32388/7U1PFG Qeios:7U1PFG

[4] T. N. Alyahya, M. E. B. Menai, and H. Mathkour. 2022. On the Structure of the Boolean Satisfiability Problem: A Survey. *ACM Comput. Surv.* 55, 3, Article 46 (2022). https://doi.org/10.1145/3491210

[5] C. Ansótegui, M. L. Bonet, and J. Levy. 2009. On the Structure of Industrial SAT Instances. In *CP*. Springer, 127–141.

[6] S. Apel, D. Batory, C. Kästner, and G. Saake. 2013. *Feature-Oriented Software Product Lines.* Springer.

[7] D. Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *SPLC*. Springer, 7–20.

[8] D. Benavides, S. Segura, and A. Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708.

[9] T. Berger and P. Collet. 2019. Usage Scenarios for a Common Feature Modeling Language. In *SPLC*. ACM, 174–181.

[10] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *VaMoS*. ACM, 7:1–7:8.

[11] T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *TSE* 39, 12 (2013), 1611–1640.

[12] A. Biere, N. Froleyks, and W. Wang. 2023. CadiBack: Extracting Backbones with CaDiCaL.

[13] T. Castro, L. Teixeira, V. Alves, S. Apel, M. Cordy, and R. Gheyi. 2021. A Formal Framework of Software Product Line Analyses. *TOSEM* 30, 3, Article 34 (2021).

[14] S. Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems.* 34–43.

[15] P. C. Clements, J. D. McGregor, and S. G. Cohen. 2005. *The Structured Intuitive Model for Product Line Economics (SIMPLE).* Technical Report. Carnegie-Mellon University.

[16] S. El-Sharkawy, A. Krafczyk, and K. Schmid. 2015. Analysing the KConfig Semantics and its Analysis Tools. In *GPCE*. ACM, 45–54.

[17] P. Franz, T. Berger, I. Fayaz, S. Nadi, and E. Groshev. 2021. ConfigFix: Interactive Configuration Conflict Resolution for the Linux Kernel. In *ICSE-SEIP*. IEEE, 91–100.

[18] J. A. Galindo, M. Acher, J. M. Tirado, C. Vidal, B. Baudry, and D. Benavides. 2016. Exploiting the Enumeration of All Feature Model Configurations: A New Perspective With Distributed Computing. In *SPLC*. ACM, 74–78.

[19] T. Heß, C. Sundermann, and T. Thüm. 2021. On the Scalability of Building Binary Decision Diagrams for Current Feature Models. In *SPLC*. ACM, 131–135.

[20] M. Janota, I. Lynce, and J. Marques-Silva. 2015. Algorithms for computing backbones of propositional formulae. *Ai Communications* 28, 2 (2015), 161–177.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21. Software Engineering Institute.

[22] C. Kästner, P. G. Giarrusso, T. Rendel, S. Erdweg, K. Ostermann, and T. Berger. 2011. Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In *OOPSLA*. ACM, 805–824.

[23] A. Kenner, C. Kästner, S. Haase, and T. Leich. 2010. TypeChef: Toward Type Checking #Ifdef Variability in C. In *FOSD*. ACM, 25–32.

[24] A. Knüppel, T. Thüm, S. Mennicke, J. Meinicke, and I. Schaefer. 2017. Is There a Mismatch Between Real-World Feature Models and Product-Line Research?. In *ESEC/FSE*. ACM, 291–302.

[25] S. Krieter, R. Arens, M. Nieke, C. Sundermann, T. Heß, T. Thüm, and C. Seidl. 2021. Incremental Construction of Modal Implication Graphs for Evolving Feature Models. In *SPLC*. ACM, 64–74.

[26] S. Krieter, R. Schröter, T. Thüm, W. Fenske, and G. Saake. 2016. Comparing Algorithms for Efficient Feature-Model Slicing. In *SPLC*. ACM, 60–64.

[27] E. Kuiter, A. Knüppel, T. Bordis, T. Runge, and I. Schaefer. 2022. Verification Strategies for Feature-Oriented Software Product Lines. In *VaMoS*. ACM, 12:1–12:9.

[28] E. Kuiter, S. Krieter, C. Sundermann, T. Thüm, and G. Saake. 2022. Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *ASE*. ACM, 110:1–110:13.

[29] J. Lee, S. Kang, and D. Lee. 2012. A Survey on Software Product Line Testing. In *SPLC*. ACM, 31–40.

[30] L. Lesoil, M. Acher, A. Blouin, and J.-M. Jézéquel. 2021. Deep Software Variability: Towards Handling Cross-Layer Configuration. In *VaMoS*. ACM, 10:1–10:8.

[31] J. H. Liang, V. Ganesh, K. Czarnecki, and V. Raman. 2015. SAT-Based Analysis of Large Real-World Feature Models Is Easy. In *SPLC*. Springer, 91–100.

[32] J. Liebig, A. von Rhein, C. Kästner, S. Apel, J. Dörre, and C. Lengauer. 2013. Scalable Analysis of Variable Software. In *ESEC/FSE*. ACM, 81–91.

[33] R. E. Lopez-Herrejon and D. Batory. 2001. A Standard Problem for Evaluating Product-Line Methodologies. In *GCSE*. Springer, 10–24.

[34] G. Masina, G. Spallitta, and R. Sebastiani. 2023. On CNF Conversion for SAT Enumeration. arXiv:2303.14971 [cs.LO]

[35] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, and S. Apel. 2016. A Comparison of 10 Sampling Algorithms for Configurable Systems. In *ICSE*. ACM, 643–654.

[36] M. Mendonça, A. Wąsowski, and K. Czarnecki. 2009. SAT-Based Analysis of Feature Models is Easy. In *SPLC*. Software Engineering Institute, 231–240.

[37] M. Mendonça, A. Wąsowski, K. Czarnecki, and D. Cowan. 2008. Efficient Compilation Techniques for Large Scale Feature Models. In *GPCE*. ACM, 13–22.

[38] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. 2001. Chaff: Engineering an Efficient SAT Solver. In *DAC*. ACM, 530–535.

[39] D.-J. Munoz, J. Oh, M. Pinto, L. Fuentes, and D. Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *SPLC*. ACM, 289–301.

[40] D.-J. Munoz, M. Pinto, L. Fuentes, and D. Batory. 2023. Transforming Numerical Feature Models into Propositional Formulas and the Universal Variability Language. *Journal of Systems and Software* 204 (2023), 111770. https://doi.org/10.1016/j.jss.2023.111770

[41] Z. Newsham, W. Lindsay, V. Ganesh, J. H. Liang, S. Fischmeister, and K. Czarnecki. 2015. SATGraf: Visualizing the Evolution of SAT Formula Structure in Solvers. In *Theory and Applications of Satisfiability Testing – SAT 2015*, Marijn Heule and Sean Weaver (Eds.). Springer International Publishing, Cham, 62–70.

[42] J. Oh, N. F. Yıldıran, J. Braha, and P. Gazzillo. 2021. Finding Broken Linux Configuration Specifications by Statically Analyzing the Kconfig Language. In *ESEC/FSE*. ACM, 893–905.

[43] L. Passos, M. Novakovic, Y. Xiong, T. Berger, K. Czarnecki, and A. Wąsowski. 2011. A Study of Non-Boolean Constraints in Variability Models of an Embedded Operating System. In *Proceedings of the 15th International Software Product Line Conference, Volume 2* (Munich, Germany) *(SPLC '11)*. ACM, New York, NY, USA, Article 2. https://doi.org/10.1145/2019136.2019139

[44] Q. Plazar, M. Acher, G. Perrouin, X. Devroey, and M. Cordy. 2019. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet?. In *ICST*. IEEE, 240–251.

[45] K. Pohl, G. Böckle, and F. J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer.

[46] R. Pohl, K. Lauenroth, and K. Pohl. 2011. A Performance Comparison of Contemporary Algorithmic Approaches for Automated Analysis Operations on Feature Models. In *ASE*. IEEE, 313–322.

[47] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *RE*. IEEE, 136–145.

[48] R. Schröter, S. Krieter, T. Thüm, F. Benduhn, and G. Saake. 2016. Feature-Model Interfaces: The Highway to Compositional Analyses of Highly-Configurable Systems. In *ICSE*. ACM, 667–678.

[49] J. Siegmund, N. Siegmund, and S. Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *ICSE*. IEEE, 9–19.

[50] C. Sundermann, T. Heß, M. Nieke, P. M. Bittner, J. M. Young, T. Thüm, and I. Schaefer. 2023. Evaluating State-of-the-Art #SAT Solvers on Industrial Configuration Spaces. *EMSE* 28 (2023).

[51] C. Sundermann, E. Kuiter, T. Heß, H. Raab, S. Krieter, and T. Thüm. 2023. On the Benefits of Knowledge Compilation for Feature-Model Analyses. (2023). Accepted.

[52] C. Sundermann, M. Nieke, P. M. Bittner, T. Heß, T. Thüm, and I. Schaefer. 2021. Applications of #SAT Solvers on Feature Models. In *VaMoS*. ACM, Article 12.

[53] C. Sundermann, H. Raab, T. Heß, T. Thüm, and I. Schaefer. 2023. *Exploiting d-DNNFs for Repetitive Counting Queries on Feature Models.* Technical Report arXiv:2303.12383. Cornell University Library.

[54] C. Sundermann, S. Vill, T. Thüm, K. Feichtinger, P. Agarwal, R. Rabiser, J. A. Galindo, and D. Benavides. 2023. UVLParser: Extending UVL with Language Levels and Conversion Strategies. In *SPLC*. ACM, 39–42.

[55] T. Thüm. 2020. A BDD for Linux? The Knowledge Compilation Challenge for Variability. In *SPLC*. ACM, Article 16.

[56] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *CSUR* 47, 1 (2014), 6:1–6:45.

[57] T. Thüm, D. Batory, and C. Kästner. 2009. Reasoning About Edits to Feature Models. In *ICSE*. IEEE, 254–264.

[58] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. 2008. SATzilla: Portfolio-Based Algorithm Selection for SAT. *JAIR* 32 (2008), 565–606.

[59] J. M. Young, P. M. Bittner, E. Walkingshaw, and T. Thüm. 2022. Variational Satisfiability Solving: Efficiently Solving Lots of Related SAT Problems. *EMSE* 28 (2022).

[60] W. Zhang, H. Zhao, and H. Mei. 2004. A Propositional Logic-Based Method for Verification of Feature Models. In *ICFEM*. Springer, 115–130.