A Performance Comparison of Contemporary Algorithmic Approaches for Automated Analysis Operations on Feature Models

Richard Pohl, Kim Lauenroth and Klaus Pohl
Paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
45127 Essen, Germany
{richard.pohl | kim.lauenroth | klaus.pohl}@paluno.uni-due.de

Abstract—The formalization of variability models (e.g., feature models) is a prerequisite for the automated analysis of these models. The efficient execution of the analysis operations depends on the selection of well-suited solver implementations. Regarding feature models, on the one hand, the formalization with Boolean expressions enables the use of SAT or BDD solvers. On the other hand, feature models can be transformed into a Constraint-Satisfaction Problem (CSP) in order to use CSP solvers for validation. This paper presents a performance comparison regarding nine contemporary high-performance solvers, three for each base problem structure (BDD, CSP, and SAT). Four operations on 90 feature models are run on each solver. The results will in turn clear the way for new improvements regarding the automatic verification of software product lines, since the efficient execution of analysis operations is essential to such automatic verification approaches.

Keywords—software product line; feature model; automated reasoning techniques; performance measurement

I. INTRODUCTION

Software product line (SPL) engineering [27] empowers companies to develop individualized software systems in a more cost-efficient way with a shorter time-to-market and at a higher quality. The variety of different products developed in a SPL must already be considered in the development process. To achieve this, variability modeling techniques are central element of product line engineering.

A. Feature Models and Automated Analysis

Feature models [22] are a common technique for modeling commonality and variability in SPLs. The basic concepts of feature models are mandatory and optional features, groups of child features and constraints. Efficient analysis operations on feature models are necessary because verification and validation techniques for software product lines often perform analysis operations directly on feature models. For example, one operation is to check a feature model for validity, i.e. to check whether a product can be derived from it. Another example is, given a set of feature models, to determine whether the feature models are semantically equivalent [11], i.e. whether they represent the same set of products. These analysis operations can reveal problems with models that may not be detected during a manual review because of the high complexity of feature models. One real example for a complex feature model is the “web shop” model from [19] with 287 features; $2.26 \cdot 10^{99}$ products can be derived from this model. Different techniques for the automated analysis of feature models have been developed during the last 20 years [10].

As described in [2], feature models can be represented by Boolean expressions to perform automated reasoning. BDD (Binary Decision Diagram) and SAT (Satisfiability) solvers can be used to analyze Boolean representations. A BDD [1] is a representation of a Boolean function as a directed, acyclic graph that is mainly used to perform analysis operations on it. BDD solvers are tools to perform such analysis operations automatically. A SAT [12] represents the question whether a Boolean expression is satisfiable, i.e. whether some assignment of values to the variables of a Boolean formula exists, such that the formula evaluates to true. SAT solvers are tools that compute a possible solution to a SAT.

An alternative approach is given in [5, 6, 7] where a feature model is transformed into a CSP (Constraint Satisfaction Problem). A CSP – in contrary to a SAT – includes constraints containing integer and interval values. A CSP solver is a tool that computes a possible solution to a CSP.

B. Problem Statement

Performing typical analysis operations, e.g. verifying variability models according to formalizations as described in [2, 11, 23], often leads to a high number of calls to solver operations in practice. Minimizing solver calls in product line analysis is an open research challenge [20]. This, in addition to the potentially high complexity of the feature model itself, requires the solver operations to be performed as efficiently as possible.

The state of the art in feature model verification presents three types of solvers: BDD solvers, CSP solvers, and SAT solvers. Nevertheless, the problems BDD, CSP, and SAT are known as NP-hard and therefore may have an exponential worst case runtime (cf. [4, 34, 12]). However, depending on the structure of the actual problem, efficient heuristics to solve these problems exist in practice. Due to the variety of implementations used to solve the different problems, the
following questions arise. First, which approach is generally more promising with regard to efficiency? Second, which approach is suitable in certain, defined situations, for example, when dealing with extraordinarily small or large feature models, or when running specific analysis operations?

According to the problem described, we define the following two research questions:

- **RQ1**: Is there a significant difference between the three approaches to transform and solve feature models (BDD-based, CSP-based and SAT-based)?
- **RQ2**: Are there specific situations, in which one approach is faster with regard to performance than the other approaches? If yes, which approach is the fastest in which situation?

C. Overview of this Paper

In order to answer the presented questions, this paper presents an experiment: nine contemporary high-performance solvers of different types (three BDD, three CSP and three SAT solvers) were used to perform four different operations on 90 feature models of real systems. Our central finding is that there is a significant influence of the choice of the solver, but none of the solver types regarded is significantly better than the others with regard to automated analysis operations on feature models in general. Nevertheless, we identified conditions under which a certain type of solver is best suited.

In Section II, we present the current state of the art and explain how our contribution is related to existing work. In Section III, we describe the plan for our experiment to evaluate the performance of different contemporary solvers. In Section IV, we describe the preparation and execution of the experiment. Section V presents the results of the experiment. In Section VI, the results are interpreted with regard to the expectations. Section VII contains the conclusion of our work and an outlook on further studies.

II. STATE OF THE ART

This section gives an overview of the state of the art regarding research on performance analysis of automated reasoning techniques using the Boolean and CSP-formalization in general and specifically, in conjunction with feature models. In addition to this, the fundamental concepts and definitions of feature model formalization are presented.

A. Related Work on the Efficiency of Problem Solving

The development of efficient heuristics to solve Boolean expressions has been a topic in research since the 1970’s. A variety of highly efficient heuristics exists. A contemporary state of the art analysis for general formal verification techniques also including techniques that use BDD, CSP and SAT solvers is given in [14]. A more exhaustive analysis only regarding SAT-solvers with an empirical evaluation is presented in [29]. Beyond this, there are well-known regular competitions like the SAT competition [18] for SAT solvers or the CPAI competition [17] for CSP solvers. The problem of all studies regarding the efficiency of problem solvers in general is that heuristics used to solve NP-hard problems are highly dependent on the problem structure and that only a subset of all possible problem structures can be regarded. For example, the SAT competition regards random problems, deliberately hard problems (e.g. n-queens problem) and industrial problems (e.g. digital circuit verification). The results of these studies are valuable for general performance studies and aim at identifying the practical limits of the considered solver technology.

B. Related Work on Feature Model Analysis

A comparison between Java-based CSP, BDD and SAT solvers was already presented by Benavides et al. in [9]. Their comparison focused on the operations “derive one product from a feature model” and “number of products”. It included the JavaBDD BDD solver, the JaCoP CSP solver and the SAT4J SAT solver. The choice of these relatively simple operations made it possible to run their tests on feature models with up to 300 features. Their feature models were random-generated. Their findings on the runtime of the “number of products” operation were similar to ours, as the BDD solver shows a lower runtime increase on larger models. Their tests include a memory usage analysis that showed an exponential growth in memory usage for the BDD solver on larger models.

In [6], a performance analysis of a CSP solver is performed based on the “all products” operation. The test was run with models containing up to 25 features and the results were generally good but with a clearly visible exponential growth, especially with highly variable models.

An extension to the FaMa framework [33], a framework for the automated analysis of feature models, with performance measurement tools is provided in [28]. With regard to performance testing, their approach aims to reveal tool-specific deficiencies by the construction of specific input models. The extension includes test data generators that support the evaluation of analysis tools for feature models, as well as tools to generate feature models that are deliberately hard to solve. Regarding the existing work, there are two remaining unsolved issues. First, there is no possibility to detect, whether performance issues detected on generated feature models are significant for the modeling of real systems. This requires a study on feature models representing real systems. The second aspect is to provide an analysis of solver performance independent of implementation aspects, like the programming language and platform, since all operations run on the FaMa Framework are based on Java implementations. Our contribution resolves both issues.

C. Fundamentals of Feature Model Formalization

This section provides a short overview of the fundamental concepts of Boolean formalization of feature models and the mapping of feature models to CSP. Moreover, notations and formal concepts used in the following sections are defined here.

Usually, feature models are trees with a root feature that is decomposed into subfeatures. There are different notations for feature models. We use the notation from [2], known as feature diagram, which is a common notation for feature models.
E. Mapping of a Feature Model to a CSP

Benavides et al. [5–7] have established a representation of a feature model as a Constraint Satisfaction Problem (CSP) [34]. For the definition of the transformation of a feature model into a CSP, see [6, 10]. For the algorithm to transform a feature model into a CSP, see [5].

As an example, the CSP formalization for the car entertainment system from Fig. 1 is given below:

| CarEntertainmentSystem = Display |
| CarEntertainmentSystem = Speakers |
| CarEntertainmentSystem = 0 ⇒ PhoneConnector = 0 |
| CarEntertainmentSystem = 0 ⇒ MediaPlayer = 0 |
| (Display = 0 ⇒ BlackWhiteLCD + ColourLCD = 0) |
| (Display > 0 ⇒ BlackWhiteLCD + ColourLCD = 1) |
| (MediaPlayer = 0 ⇒ CDPayer + DVDPlayer + MP3Player = 0) |
| (DVPlayer > 0 ⇒ ColourLCD > 0) |
| (DVPlayer > 0 ⇒ PhoneConnector = 0). |

F. Analysis Operations on Feature Models

Among others, the following operations can be performed to analyze feature models:

- Number of products (cardinal): determines how many valid products can be derived from a feature model. The number of potential products for a feature model \( M \) is also called \( \text{cardinal}(M) \) [6]. For the example from Fig. 1, this value is 20.
- The variability specified in a feature model defines the number of products which can be derived from a product line. At one extreme, only one product can be derived. This is the feature model with the lowest variability. At the other extreme, regarding a specific feature model \( M \), the theoretically highest variability would be given by a feature model \( M' \) which contains only the leaf features of \( M \) and no relationships (requires, excludes, alternative group constraints, etc.) among them. Based on this, the variability factor [6] is defined:

\[
VF(M) = \frac{\text{cardinal}(M)}{\text{cardinal}(M')}
\]

- All products: determines all products that can be derived from a feature model.
- Valid: a feature model \( M \) is valid, if there are valid products that can be derived from it, i.e. \( \text{cardinal}(M) > 0 \). A feature model containing two mandatory features \( f_1 \) and \( f_2 \), where \( f_1 \) excludes \( f_2 \) is an example for an invalid or void feature model.

More advanced operations include additional input parameters, e.g. other feature models, products or product configurations; see [8, 10].

III. EXPERIMENT PLANNING

In this section, we describe the performance comparison between nine implementations regarding basic operations on feature models. The comparison is planned, executed and analyzed as an experiment following the guidelines from [36].
A. Selection of Example Models

The intended test set includes feature models from SPLOT [19], a generally accepted source for example feature models that is used extensively for research work. At the time of our study, SPLOT contained 101 models, 90 of them were used for our test.

A pre-test was run with all 101 models to check the compatibility with the solvers. Some models had to be excluded from the test because the valid operation was cancelled on at least one of the solvers after three hours. In summary, 90 models were selected for the test, including the “web shop” model – the largest feature model available from SPLOT (287 features, cardinal $= 2.26 \cdot 10^{49}$). The “valid” operation was run on all models; the other operations were performed on subsets on which a termination within three hours could be expected on most of the solvers.

B. Selection of Operations

During the test, four operations were performed on feature models, including the “number of products” operation, the “variability” operation, the “all products” operation and the “valid” operation. These four operations are basic operations that represent different categories of operations with different degrees of complexity. The “valid” operation is a simple satisfiability check on a feature model. Performing the “number of products” operation, products are counted, i.e. satisfying variable assignments to the formalization of a feature model. The “variability” operation additionally considers relations among features, because different types of features – leaf features and inner features – have to be considered to calculate the result. “All products” is an operation that computes all valid products. This operation is a more complex operation which tends to have longer runtimes and even becomes infeasible on larger feature models [8].

C. Selection of Solvers

Solvers representing the current state of the art were selected for the test. As a first step, three solvers used in the FaMa framework [33] were selected for the experiment. FaMa is a contemporary framework for the automated analysis of feature models. The FaMa framework uses the SAT4J SAT solver [24], the Choco CSP library [21] and the JavaBDD BDD library [35]. The other two CSP-solvers selected for the test are Cream [32] and Mistral [16]. Mistral won in 8 of 21 categories in the CSP 2009 Competition [17] – a well-recognized competition for CSP solvers. Regarding SAT solvers, there is a huge variety of solvers. A well-recognized competition for SAT solving is the SAT competition [18] that takes place every two years. Two different solvers were selected from the competitions in 2007 and 2009 that won gold medals in two different categories: PicoSAT [3] and clasp [15]. PicoSAT is a highly optimized solver that won a medal in solving “industrial” problems; it won its medal in the category of “random” SAT instances. For BDD solving, we included Buddy [25] and Cudd [30], two BDD-libraries that are widely used in many research works and deal with performance analysis on BDDs.

Shortly summarized, we selected three BDD, three CSP and three SAT solvers. From these nine solvers, four are written in Java: SAT4J, Choco, JavaBDD and Cream. The other five solvers are written in C or C++. Influences of the programming language are considered in the analysis of the experiment.

D. Experiment Design

The dependent variable of our experiment is the runtime of the solver. The independent variables are (1) the solver used, in terms of (1a) the solver problem structure (i.e. BDD, CSP, SAT) and (1b) the solver platform (i.e. native or Java), (2) the operation run on the solver, where we always regarded solvers or operations in combination with the solvers used, and (3) the cardinal value of the model as a measure for model size.

The effects of independent variables are determined by separating the runtime measurements into groups, e.g. runs from BDD, CSP or SAT solvers, etc. To determine the effects of the model size on the runtimes, the models were divided into three groups, resulting in a proportional experiment design. Models with cardinal up to 100 form the small group, models with cardinal $> 100,000$ form the “large” group. All models in between form the “medium” group. At this point, a prerequisite for this approach was to assume that cardinal is a representative measure for the size of feature models, with “large” models being models that tend to have higher runtimes during analysis operations. To confirm this, we aim at rejecting the hypotheses that high runtimes are not correlated to high cardinal values.

In summary, we define the null hypotheses $H_0$:

a. The mean runtimes will be equally distributed among the tested solvers.
b. The mean runtimes will be equally distributed among the different tested solver problem structures (i.e. BDD, CSP and SAT).
c. The mean runtimes will be equally distributed among the different tested solver platforms (i.e. native, Java).
d. There will be no interaction between solver and operation performed.
e. There will be no significant correlation between cardinal and the runtime of a model.
f. There will be no interaction between solver by operation performed and model size.

Our experiment aims at rejecting $H_0$, which helps to resolve the defined research questions.

IV. Experiment Operation

This section provides an overview of the actions taken to prepare the environment and to run the experiment. The experiment was run on an Intel Core2Duo 6550 machine with a Suse Linux 32-bit operating system and 2.00GB of RAM. For the Java-based solvers, the Java virtual machine was used with an initial heap size of 1.00 GB and a maximum heap size of 2.00 GB. Each solver performed one operation and was then restarted to guarantee equal conditions for each operation. Moreover, the solver was changed after each run to equalize technical influences that may occur during a short period of time. Five runs for each operation were performed on each model to minimize random deviations from the system.
A. Preparation of Solvers

To run a specific operation on a solver, the feature models and the operations need to be mapped to the solver-specific problem structures that are BDD, CSP or SAT. We provide a straight-ahead implementation of the known mappings from current research work. Our implementation transforms the feature model into the solver problem input language and calls the solver to compute the desired solution. Moreover, it measures the time between the beginning of the transformation into the solver-specific problem and the termination of the solver operation. All native code was compiled using the Gnu C/C++ compilers with the highest optimization level (O3).

B. Input Format

We used the SPLOT XML format as input for our test. For the native implementations, we developed a parser for this format in C. The Java implementations were used in connection with the Java parser provided by SPLOT. All SAT solvers are compatible with a special exchange format for Boolean expressions in conjunctive normal form (CNF) – the Dimacs CNF format. Therefore, this format, extended with the information of the number of leaf features needed for the variability operation, was used as input for the SAT solvers. A CNF was generated from each model using a Java implementation based on the Java parser provided by SPLOT. The performance of our Java implementation was very low compared to the runtimes of the native C solvers. Our expectation was that an optimized native implementation would show much better runtimes. For this reason, adding these times to the results of the operations would destroy the solver-specific distributions for these solvers. The time for generating a CNF varied between .2 ms and 396 ms, increasing on larger models. There is a positive correlation of .82 between the number of features and the conversion time; both are significant at the 1% significance level. This is an indication of a linear increase of this time with larger models.

C. Results from the “Variability” Operation

The variability operation was run on a set of 86 models. The operation is quite similar to the “number of products” operation and all solvers show a similar runtime distribution. Because of the number of only three results – all from the SAT4J solver – observed in [1,800s; 10,800s] on the “number of products” operation, runs longer than 1,800 seconds were cancelled on the regular “variability” operation. These were all runs with the 13 highest-cardinal models on the PicoSAT and SAT4J solvers and all runs of the 7 highest-cardinal models on all non-BDD solvers. On smaller models up to a cardinal value of at least 55,681,560 were cancelled on all except the BDD solvers – which terminated on all models within 6.1 s. Some noticeable runtime peaks up to 10 seconds appear on the BDD solvers with no obvious relation to cardinal.

D. Results from the “Number of Products” Operation

The “number of products” operation was run on a set of 86 models. The operation is quite similar to the “number of products” operation and all solvers show a similar runtime distribution. Because of the number of only three results – all from the SAT4J solver – observed in [1,800s; 10,800s] on the “number of products” operation, runs longer than 1,800 seconds were cancelled on the regular “variability” operation. These were all runs with the 13 highest-cardinal models on the PicoSAT and SAT4J solvers and all runs of the 7 highest-cardinal models on all non-BDD solvers. On smaller models up to a cardinal value of 1,000, the native SAT and CSP solvers produce the best runtimes. The Java based SAT and CSP solvers produce the longest runtimes on these models. On larger models, the runtimes of the SAT and CSP solvers show a strong increase. On all models with cardinal > 21,984, all BDD solvers were faster than all other solvers.
<table>
<thead>
<tr>
<th>Operation</th>
<th>Solver</th>
<th>Type / Language</th>
<th>Buddy</th>
<th>Cudd</th>
<th>JavaBDD</th>
<th>Choco</th>
<th>Cream</th>
<th>Mistral</th>
<th>clasp</th>
<th>PicoSAT</th>
<th>SAT4J</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BDD / C</td>
<td>.00257</td>
<td>.00173</td>
<td>.00253</td>
<td>.14638</td>
<td>.004214</td>
<td>.00032</td>
<td>.00004</td>
<td>.000038</td>
<td>.031362</td>
</tr>
<tr>
<td>Valid</td>
<td></td>
<td>Maximum</td>
<td>5.8421266</td>
<td>3.655897</td>
<td>23.794145</td>
<td>216336</td>
<td>0.67948</td>
<td>.01273</td>
<td>.00076</td>
<td>.000271</td>
<td>.045895</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>.0682737</td>
<td>.045693</td>
<td>.296625</td>
<td>1.61092</td>
<td>.01099</td>
<td>.00199</td>
<td>.00065</td>
<td>.000664</td>
<td>.035454</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard Dev.</td>
<td>61.97979</td>
<td>38.508</td>
<td>2.599787</td>
<td>0.08964</td>
<td>0.00435</td>
<td>.00168</td>
<td>.00004</td>
<td>.000032</td>
<td>.00168</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Small Models</td>
<td>.000292</td>
<td>.003325</td>
<td>.003631</td>
<td>.157392</td>
<td>.066808</td>
<td>.00113</td>
<td>.00043</td>
<td>.000049</td>
<td>.034840</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Medium Models</td>
<td>.000518</td>
<td>.003532</td>
<td>.015898</td>
<td>.160453</td>
<td>.009652</td>
<td>.00189</td>
<td>.00044</td>
<td>.000060</td>
<td>.035428</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Large Models</td>
<td>278286</td>
<td>1.76347</td>
<td>1.185696</td>
<td>.168089</td>
<td>.019830</td>
<td>.00318</td>
<td>.00049</td>
<td>.000093</td>
<td>.036485</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cancelled</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>#Products</td>
<td>Minimum</td>
<td>.000273</td>
<td>0.00222</td>
<td>.002694</td>
<td>.148733</td>
<td>.00211</td>
<td>.000032</td>
<td>.000053</td>
<td>.000046</td>
<td>.033381</td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td>Maximum</td>
<td>6.093089</td>
<td>3.657715</td>
<td>24.082592</td>
<td>262.429446</td>
<td>1682.205184</td>
<td>255.782832</td>
<td>266.273856</td>
<td>9644.66142</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>.073562</td>
<td>.047573</td>
<td>.306905</td>
<td>5.52992</td>
<td>33.491217</td>
<td>5.575607</td>
<td>.374909</td>
<td>4.391224</td>
<td>282.872697</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard Dev.</td>
<td>6.53388</td>
<td>.391855</td>
<td>2.583879</td>
<td>32.208616</td>
<td>207.642737</td>
<td>32.635831</td>
<td>2.195299</td>
<td>31.478829</td>
<td>1489.127514</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Small Models</td>
<td>.003045</td>
<td>.003358</td>
<td>.008302</td>
<td>.161222</td>
<td>.014490</td>
<td>.002016</td>
<td>.000057</td>
<td>.000031</td>
<td>.000045</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium Models</td>
<td>.000553</td>
<td>.002594</td>
<td>.016343</td>
<td>.270658</td>
<td>.135026</td>
<td>.006016</td>
<td>.000508</td>
<td>.007425</td>
<td>.270813</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Large Models</td>
<td>335363</td>
<td>2.03537</td>
<td>1.382935</td>
<td>35.661492</td>
<td>222.871245</td>
<td>37.304022</td>
<td>2.497871</td>
<td>63.634401</td>
<td>2205.376000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancelled</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>14</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Variability</td>
<td>Minimum</td>
<td>.000278</td>
<td>.003169</td>
<td>.002717</td>
<td>.147733</td>
<td>.002469</td>
<td>.000035</td>
<td>.000043</td>
<td>.000046</td>
<td>.047012</td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td>Maximum</td>
<td>6.077616</td>
<td>3.700259</td>
<td>23.93766</td>
<td>124.139312</td>
<td>808.174803</td>
<td>145.394928</td>
<td>11.028465</td>
<td>266.638031</td>
<td>18.703533</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>.074040</td>
<td>.047012</td>
<td>.395484</td>
<td>2.291598</td>
<td>12.590819</td>
<td>2.496575</td>
<td>1.732399</td>
<td>4.393100</td>
<td>66.6761</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard Dev.</td>
<td>6.55248</td>
<td>.398580</td>
<td>2.582086</td>
<td>14.051255</td>
<td>91.252882</td>
<td>16.476562</td>
<td>1.245492</td>
<td>31.472471</td>
<td>2.484362</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Small Models</td>
<td>.003100</td>
<td>.003502</td>
<td>.008375</td>
<td>.161485</td>
<td>.014599</td>
<td>.002023</td>
<td>.000058</td>
<td>.000031</td>
<td>.000058</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium Models</td>
<td>.000537</td>
<td>.003707</td>
<td>.015978</td>
<td>.270387</td>
<td>.136224</td>
<td>.005934</td>
<td>.000507</td>
<td>.007425</td>
<td>.258503</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Large Models</td>
<td>339603</td>
<td>2.11041</td>
<td>1.368174</td>
<td>15.42764</td>
<td>89.980902</td>
<td>17.260885</td>
<td>1.259767</td>
<td>63.366767</td>
<td>74.86834</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancelled</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>Minimum</td>
<td>.000277</td>
<td>.002943</td>
<td>.003027</td>
<td>.148874</td>
<td>.004314</td>
<td>.003027</td>
<td>.000062</td>
<td>.000054</td>
<td>.033228</td>
<td></td>
</tr>
<tr>
<td>Products</td>
<td>Maximum</td>
<td>1.175714</td>
<td>.040904</td>
<td>15.537384</td>
<td>47.241702</td>
<td>11.5732856</td>
<td>15.537384</td>
<td>5.353888</td>
<td>277.46176</td>
<td>9595.338719</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>.043589</td>
<td>.026188</td>
<td>.626260</td>
<td>1.997535</td>
<td>4.132795</td>
<td>1.085581</td>
<td>24540</td>
<td>4.538322</td>
<td>281.21465</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard Dev.</td>
<td>1.91142</td>
<td>.091499</td>
<td>2.611325</td>
<td>7.378875</td>
<td>17.833629</td>
<td>4.614967</td>
<td>1.039977</td>
<td>32.713334</td>
<td>1482.222646</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Small Models</td>
<td>.000312</td>
<td>.003137</td>
<td>.004721</td>
<td>.163364</td>
<td>.017826</td>
<td>.000321</td>
<td>.000106</td>
<td>.000031</td>
<td>.000045</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium Models</td>
<td>.001136</td>
<td>.006337</td>
<td>.052878</td>
<td>.341861</td>
<td>191.523</td>
<td>0.01161</td>
<td>.002935</td>
<td>.004460</td>
<td>.000160</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Large Models</td>
<td>327078</td>
<td>1.72696</td>
<td>4.882010</td>
<td>13.898531</td>
<td>31.56034</td>
<td>8.431074</td>
<td>1.90500</td>
<td>65.779775</td>
<td>2193.121810</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancelled</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

“Small models” is the average of models with cardinal ≤ 10^2 (36 models), “Medium models” is the average of models with cardinal ∈ (10^2, 10^5] (32 models), “Large models” is the average of models with cardinal > 10^5 (22 models on “valid”, 19 on “cardinal”, 18 on “variability”, 10 on “all products”), “Cancelled” refers to cancellations after a runtime of 3 hours (5 hours on variability operation).
The Buddy BDD solver was always the fastest BDD solver with one exception – Cudd wins the race on the “web shop” model. Regarding the results of the low performers, on small models, Choco, the Java CSP solver, is the solver with the longest runtimes. Though, the runtime increase of Choco on high cardinal models is relatively low. On models with a cardinal value above 1,000,000, the runtimes of Choco and the Mistral native CSP solver are comparable.

D. Results from the “All Products” Operation

Fig. 3 shows the average runtimes of the “all products” operation. On this operation, the clasp SAT solver is fastest on small models with cardinal ≤ 200. Above this threshold, the Buddy BDD solver is the fastest solver up to cardinal ≤ 7056. On all larger models, the Cudd BDD solver is the fastest solver with Buddy on the second place. Up to a cardinal value of 16,384, the clasp SAT solver is faster than at least one BDD solver. The Choco CSP solver behaves similar to the cardinal operation; its increase in runtime is less strong than the increase on other SAT and CSP solvers. Another observation is that the PicoSAT solver did not terminate within reasonable time on the 5 models with the highest cardinal value. The JavaBDD solver showed high deviations that occur independently from cardinal.

The standard deviations of the runtimes for each solver ranked from .09 to 1644.6. The native BDD solvers produced runtimes with low standard deviations between .09 (Cudd) and .19 (Buddy). The other solvers, except for the SAT4J solver, produced standard deviations up to 32.7. The SAT4J SAT solver produced a standard deviation of 1482.2.

As explained in [8], the “all products” operation is often infeasible on larger models due to the high memory consumption of the operation. We restricted the operation test set to 78 models with a cardinal value of 3,000,000 or less.

E. General Significance of Runtime Measurements

The analysis of variance (ANOVA) is a statistical standard procedure for analyzing effects of independent variables on a dependent variable with regard to a certain hypothesis. ANOVA can be applied in situations of independent cases with a normal distribution of the residuals and homogeneity of variances; it is tolerant to slight departures from these preconditions (cf. [31]). In our case, independence and homogeneity of variances is achieved through the experiment design. We assume a normal distribution of the residuals from the high number of real models. Moreover we assume that the collected models are representative. In the experiment a 5 x 4 x 9 repeated measures two-way ANOVA was conducted. The goal was to determine whether the choice of the operation and the solver influenced the runtimes of the five repeated measures. The results indicated a main effect for the choice of the solver with $F(8, 560) = 20.424, p < .05$. Moreover, a main effect for the solver by operation was indicated with $F(24, 1680)= 20.491, p < .05$. On the “valid” operation, the clasp SAT solver had the shortest runtime ($M = 4.51 \cdot 10^{-5}$, SD = $1.394 \cdot 10^{-6}$). Regarding the “number of products” operation, the Cudd BDD solver had the shortest runtime ($M = .0476$, SD = .3919). On the “variability” operation, the Cudd BDD solver had the shortest runtime ($M = .0470$, SD = .3986). On the “all products” operation, the Cudd BDD solver had the shortest runtime ($M = .0262$, SD = .0915).

F. Significance of Problem Structure and Platform

A 5 repeated measures two way ANOVA was conducted to determine whether the solver problem structure (BDD, CSP or SAT) had a significant influence on the runtimes. The result of $F(2, 2947) = 5.788, p < .05$ indicates a signifi-
significant influence of the solver problem structure. Though, regarding the solver problem structure by operation, a result of \( F(6, 2947) = 1.991, p = .064 > .05 \), indicates no main effect at the 5% significance level.

A second ANOVA was conducted to determine whether there is an effect of the solver platform (native or Java) on the runtimes. The result of \( F(1, 2951) = 8.460, p < .05 \) indicates a main effect. Regarding the platform by operation, a result of \( F(3, 2951) = 2.640, p < .05 \) indicates a main effect.

G. Correlation Analysis

To determine influencing factors to runtime, some complexity measures were calculated for all test models. The set of complexity measures includes the number of features, cardinal, the Extra Constraint Representativeness (ECR) [26] and two well-known complexity measures from graph theory: the cyclomatic number and the number of cliques. Pearson’s correlation coefficient between each of the complexity measures and the runtime measurements was calculated and tested for significance at the 1% significance level.

Significant correlations were found for all solvers between the number of features and between cardinal and the mean runtimes. The correlation between cardinal and the mean runtimes were extraordinarily high. For the “all products”, “number of products” and “variability” operation, the correlation coefficients rank from \[.65, .953\], whereby all BDD solvers showed high-mean runtimes were extraordinarily high. For the “all products” operation, only the BDD solvers, with the correlation coefficients in \[.365, .860\]. The significant values are in \[.369, .376\]. Considering the “variability” operation shows a higher correlation between the number of features and the mean runtime on all non-BDD based solvers, ranging from \[.65\] to \[.953\], whereby all BDD solvers showed higher correlations between cardinal and the mean runtime on the "valid" operation.

There was no significant correlation between ECR and mean runtime for any solver-operation combination.

A significant correlation between cyclomatic number and the mean runtime was found on the "valid" operation for all solvers, with the correlation coefficients in \[.365, .860\]. The three CSP solvers produced the three highest correlation coefficients in \[.775, .860\]. Considering the “variability” operation, the correlation between cyclomatic number and mean solver runtime is significant for all but the SAT4J and PicoSat solvers. The significant values are in \[.369, .376\]. On the “number of products” operation, only the BDD solvers correlation coefficients in \[.369, .378\] are significant.

The correlation coefficient between the number of cliques and the mean runtime was significant for all solvers on the “valid” operation with correlation coefficients in \[.365, .655\] and solely the correlation coefficients of the BDD solvers in \[.365, .366\]. On the “number of products” and “variability” operation, significant correlation coefficients were found for all solvers except SAT4J and PicoSat. The correlation coefficients for these operations are within \[.353, .506\], the correlation coefficients for the BDD solvers on both operations within \[.366, .368\].

H. Significance of Model Size

The initial 5 x 4 x 9 repeated measures two-way ANOVA indicated a main effect of model size with \( F(2, 70) = 14.606, p < .05 \), regarding the division into three groups, as mentioned above. Therefore, in addition, one ANOVA was conducted for each of the three groups to determine whether there are significant influences within these groups. On the small models group, \( F(8, 280) = 13655.016, p < .05 \) indicates a main effect for the choice of solver, \( F(24, 840) = 142.445, p < .05 \) indicates a main effect for solver by operation. Regarding the medium models group, \( F(8, 248) = 78.178, p < .05 \) indicates a main effect for the choice of solver, \( F(24, 744) = 36.115, p < .05 \) indicates a main effect for solver by operation. On the large models group, \( F(8, 32) = 1.513, p = .192 > .05 \) indicates no main effect for the choice of solver. Moreover, \( F(24, 96) = 1.515, p = .081 > .05 \) indicates no main effect for solver by operation. We assume that this effect results from the low number of results considered due to the high number of cancellations in this group, especially on complex operations, where some solvers delivered only 5 results. Therefore, we repeated the ANOVA considering the results of only the “valid” and “number of products” operation and the 8 solvers that produced at least 10 on these operations. The results of the repeated analysis indicated a main effect of solver with \( F(7, 63) = 3.407, p < .05 \) (solver by operation with the same results, because only two operations were tested).

For the descriptive statistics of the three groups, see Table I.

VII. INTERPRETATION OF RESULTS

This section provides an interpretation of the results based on the hypotheses defined in the experiment design. The interpretation provides an answer to our two research questions. In general, we can reject \( H_0 \) at the 5% significance level, leaving only a small chance of error. In Subsection A, we answer RQ1. In the following Subsections B and C we regard influences on the solver performance to resolve RQ2. Subsection D presents threats to validity.

A. Choice of Solver

Based on the results from the three ANOVAs performed to determine the significance of influences to the runtime, we reject part a–d of the null hypothesis and conclude that there is a significant influence of (a) the solver in general, (b) the solver problem structure and (c) the solver platform on the runtime, partially depending on the operation performed. Therefore, we can conclude that there is a significant difference between the runtimes of BDD, CSP and SAT solvers.

B. Specific Influences on the Solver Performance

The result of the operation having an influence slightly below the significance level in the “large models” group may be caused by the similarities in the runtime distributions of the variability and cardinal operations and by excluding some very large models due to runtimes above 3 hours. This leads to a smaller number of results in the “large models” group. The mean runtime of BDD solvers is significantly shorter than the mean runtime of CSP solvers, which itself is significantly shorter than the mean runtime of SAT solvers. The mean runtime of the native solvers is significantly shorter than the mean runtime of the Java-based solvers.
Regarding the four operations, the BDD solvers show a higher standard deviation than the native CSP and SAT solvers on the least complex “valid” operation. While the runtimes of the native CSP and SAT solvers are very stable, the runtimes of the BDD solvers are less predictable depending on the cardinal value of the model.

The BDD solvers in the test produced the lowest average runtime on large models on all, except for the “valid” operation, on which they were outperformed by the native SAT solvers. On all more complex operations, the BDD solvers showed the lowest runtimes at least within the “large models” group. Small to medium models were solved by a native SAT solver (clasp) in the shortest time. Though, the overall maximum runtime of the BDD solvers during the test was below one minute.

The correlation analysis indicates a correlation between runtime and several graph complexity measures, depending on solver and operation. Furthermore, the BDD solvers showed a different behavior in the correlation analysis compared to SAT and CSP solvers regarding the “valid” operation. The results of the correlation analysis, as well as the peaks in runtime in conjunction with the high standard deviations of the BDD solvers lead us to assume that there is a more complex relation between specific properties of feature models and the runtime of the BDD solver.

Considering the model size, we can reject parts e–f of the null hypotheses. The ANOVA performed on the three groups of different-sized models indicates a significant influence of the model size. Beyond this, the results from the correlation analysis indicate a significant correlation between cardinal and runtime that is higher than the correlation between any other tested factor and runtime. This confirms that the choice to form categories of models based on cardinal is reasonable.

C. Threats to Validity

The validity of an experiment can be expressed in terms of internal and external validity [36], where the internal validity refers to possible confounders affecting independent variables without the researcher’s knowledge, e.g. factors in the environment affecting the performance of one solver. Although the experiment was designed to minimize these factors, one threat to the internal validity remains. As discussed above, we used different input formats for the BDD, CSP and SAT solvers. While the BDD and CSP solvers use the SPLIT XML format, the SAT solvers process a Dimacs CNF. An implementation of the SAT solvers solely based on the SPLIT XML format may show slightly different results. This may be a factor when comparing high-performance SAT and CSP solvers in situations with similar runtimes, which particularly occur between the two SAT solvers clasp and PicoSAT and the CSP solver Missat. This does not affect the Java-based solvers, because the runtimes of our Java-based implementation of a format converter are well below the minimum runtime of the Java-based solvers.

The external validity refers to the generalizability of the results. For example, it may be possible that some real feature models contain modeling constructs that are not considered in the models obtained from [19], but affect performance. We considered a large number of feature models to minimize this factor. Beyond this, there are two aspects of the solver implementations. First, the memory structure for the results of the “all products” operation can be implemented in a more or less efficient way. Second, the transformation algorithms for transforming a feature model into a BDD or a CSP can be more or less efficient.

VIII. Discussion and Conclusion

A. Conclusion from the Results

The results of the experiment indicate that different solvers can be superior with regard to performance on specific models or performing specific operations, with the BDD solvers producing the best results in most situations. In particular, the BDD solvers produced the best performance in the case of larger models, as well for more complex operations which generally tended to produce longer runtimes. Native implementations of CSP and SAT solvers, like Missat, clasp and PicoSAT, produced superior results either on small models or on the relatively simple “valid” operation. Moreover, native CSP and SAT solvers produced relatively low standard deviations on this operation. This makes their runtimes more predictable compared to the BDD solvers with noticeable runtime peaks even for simple operations and small models. CSP and SAT solvers are especially a good choice for smaller models in a time-critical environment, but a potentially high number of solver calls.

When analyzing larger feature models, typically advanced analysis operations (e.g. the detection of dead features) consist of a large number of calls of basic operations. For example, in an interactive modeling environment with automatically-triggered checks that are performed to give immediate feedback to the modeler, this can lead to a time-critical situation. In this case, the differences between native CSP and SAT solvers on the one hand and BDD solvers on the other hand are particularly considerable.

On the “valid” operation, for BDD solvers, the correlation between the actual cardinal value and runtime was higher, while for non-BDD solvers, the correlation between the number of features and runtime was higher. The maximum value for cardinal in a feature model with \( n \) features is \( 2^n - 1 \). This suggests an exponential runtime increase with \( n \) for non-BDD solvers on the “valid” operation.

B. Summary and Outlook

We performed an experiment considering four analysis operations. These operations are representative for different types of operations typically performed on feature models. We described statistically significant effects on the runtime of different solver implementations that use different base problem structures and platforms. The outcome was that Java implementations produced significantly higher runtimes than native implementations.

Our central finding is that neither the Boolean nor the CSP-representation is superior in all situations with regard to the runtime of the corresponding BDD, SAT or CSP solvers. The results confirm the findings from related work by showing that BDD solvers outperform other solvers on large models – even on real instead of generated models. Compared to
general studies on the efficiency of problem solving techniques, the results of our experiment show that for feature model analysis, the potentially exponential growth of the BDD size is not a noticeable problem for the practical feasibility of the analysis compared to the runtime increase experienced by CSP and SAT solvers. In addition, the experiment showed that CSP and SAT solvers in general behave similar, but the solver implementation can lead to a significant difference, especially between native and Java implementations. In general, the results of our study facilitate the choice of the appropriate solver depending on the size of the models to be analyzed and the intended analysis operation.

We envision further studies to analyze the applicability of the representations and solvers in a specific context. In particular, we plan to apply advanced analysis techniques from graph theory to the models in order to determine further possible influences on the performance results and to investigate the cause of the correlations found. As a long-term objective, we plan to include our findings in the development of a tool framework for the automated analysis of variability models to support the automated selection of suitable solvers.

ACKNOWLEDGMENT

This work has been funded by the DFG under grant PO 607/4-1 KOPI.

REFERENCES