# Pick Me: Judging Sample Quality with Binary Decision Diagrams

Tobias Heß
University of Ulm
Germany

Aaron Molt
University of Ulm
Germany

Sabrina Böhm
University of Ulm
Germany

Sebastian Krieter
TU Braunschweig
Germany

Thomas Thüm
TU Braunschweig
Germany

## ABSTRACT

Faults not arising from individual but the interaction of multiple components challenge the testing of software products. This problem is escalated by configurable systems which, in addition to a test, also require a configuration including the interaction to test. T-wise sampling techniques generate a set of configurations such that any interaction between $t$ features appears in at least one configuration. However, trade-offs need to be made when applying $t$-wise sampling in practice, as time and resources for testing are limited. For instance, even for $t = 2$, some real-world systems require thousands of configurations to cover all interactions, an infeasible sample size in practice. To address these challenges, multiple parameterized and randomized samplers have been developed in the past, raising the question "which sample to use for testing?". In this work, we harness recent advances in the compilation of binary decision diagrams (BDD) and propose **pm**, a BDD-based metric for comparing the quality of samples. Our evaluation demonstrates that **pm** is easy to compute and significantly acts as a proxy metric for both sample size and 2-wise coverage. Furthermore, we present an exact algorithm to count 2-wise interactions on BDDs that also scales to the infamous automotive02v4 model, from SPLC's sampling challenge.

## KEYWORDS

T-Wise Sampling, Sampling Quality, Combinatorial Interaction Testing, Feature Models, Binary Decision Diagrams

**ACM Reference Format:**
Tobias Heß, Aaron Molt, Sabrina Böhm, Sebastian Krieter, and Thomas Thüm. 2025. Pick Me: Judging Sample Quality with Binary Decision Diagrams. In *Proceedings of 29th International Systems and Software Product Line Conference (SPLC'25).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Testing software systems is a challenging, high-cost, low-reward, but essential task in software engineering [10]. For configurable systems, two additional challenges are added. First, faults commonly arise from the interaction of multiple features [1, 12, 25]. Second, the often unfathomably large number of configurations [53] makes testing all configurations infeasible [25, 28, 41]. Therefore, one first has to find a suitable set of configurations of manageable size, before configurable systems may be tested.

In the case of fault detection through interaction testing, such sets are typically generated by means of t-wise samplers. These generate configurations with the goal of covering any $t$-wise interaction with at least one configuration. A set of configurations covering all such interactions is called a sample with 100 % $t$-wise coverage [36, 46, 58].

While previous works recommended $t \geq 3$ for effective interaction testing [25, 41], Heß et al. demonstrated that only $t = 2$ is feasible for many real-world systems [28]. However, even for $t = 2$, some systems still required thousands of configurations [28, 36, 37] or hours of sampling time [28, 35, 36]. As a system's test suite is commonly executed for each configuration [25], but resources and time for testing are almost always limited [25, 45], trade-offs need to be made [10].

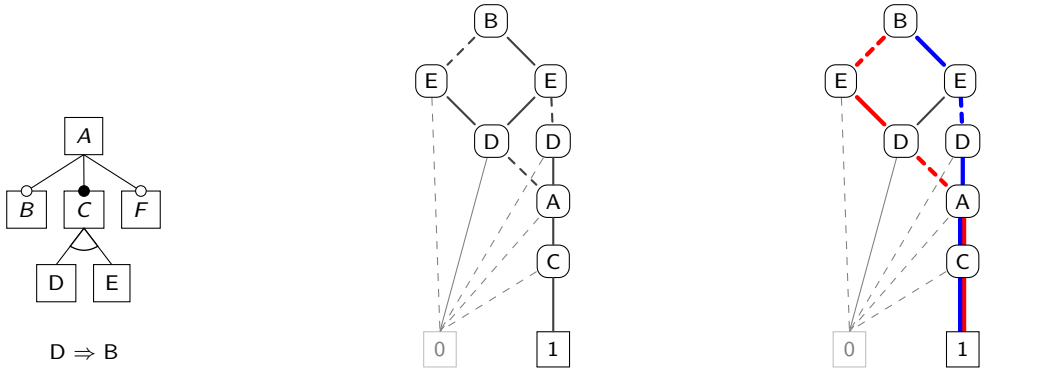
Clearly, smaller samples are desirable, but is it worth to invest potentially orders of magnitude more time into sampling to reduce the sample size? While higher coverage is desirable, when does the added expense for testing additional test cases negate the benefits of additional coverage? With multiple samples available, which sample to use? Naturally, these questions will be answered diversely depending on the system, the application, and extrinsic factors, such as available resources for testing [25].

Even when these questions can be answered, the challenge of computing the quality of a sample remains. For many real-world systems, both the numbers of total and covered 2-wise interactions can be computed efficiently. This is, however, not the case for larger systems, such as automotive02v4 [46]. With more than half a billion of valid 2-wise interactions, state-of-the-art tools like YASA [36] currently require hours and 16+ GB of RAM to count the interactions in automotive02v4. And even when these numbers or their quotient, the coverage, are known, it remains unclear how to weigh them against other factors, such as sample size or uniquely covered interactions.

In this work, we address this issue by proposing a novel metric for judging the quality of samples. Our pick-me (**pm**) metric acts significantly as a proxy metric for both sample size and coverage, but can be computed efficiently, even for automotive02v4. Its computation harnesses recent advances in the compilation of binary decision diagrams (BDD) [19, 29] and, in essence, marking and counting BDD edges. As a byproduct, we also present an exact algorithm for generating and counting 2-wise interactions on BDDs. With this algorithm, we are able to count the number of 2-wise interactions


*SPLC'25, September 01–September 05, 2025, A Coruña, Spain*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn




**(a) Feature Model $M$** **(b) Binary Decision Diagram $BDD(M)$** **(c) Configurations as BDD Paths**

**Figure 1: Running examples of a feature model $M$, a BDD encoding $M$, and configurations of $M$ as 1-paths in the BDD**

in automotive02v4 in less than 3 minutes, including preprocessing and BDD compilation. Thereby, we close the scalability gap of the current state of the art for all feature models for which BDD compilation is feasible but SAT-solving-based interaction counting is hard.

The remainder of this work is structured as follows. In Section 2, we introduce feature models, their translation to propositional logic and binary decision diagrams, and give an introduction to t-wise sampling. In Section 3, we give an overview of the state of the art and motivate the utility of this work. In Section 4, we present the foundational idea of our approach as well as the algorithm for counting 2-wise interactions of BDDs and foremost, the **pm** metric. In Section 5, we evaluate our approach on a number of real-world feature models and demonstrate the utility of our metric. Finally, we discuss alternative approaches to testing configurable systems in Section 6 and conclude the work in Section 7.

## 2 BACKGROUND

In this, work we compile binary decision diagrams from feature models, in order to judge the quality of 2-wise samples without requiring to compute their coverages. In the following, we give a brief overview of feature models, binary decision diagrams, and t-wise sampling.

### 2.1 Feature Models

Feature models have emerged as the standard for modeling and reasoning about the variability in configurable systems [7, 8]. Configurable options are modeled as *features* that can be selected or deselected [21]. We will only consider Boolean features in this work.

Figure 1a depicts the common visual representation of a feature model, in which the hierarchy between features is given by a tree-like diagram [7]. A selected child feature always requires the selection of its parent feature. Vice versa, mandatory child features (●, like $C$) must be and optional child features (○, like $B$ or $F$) may be selected when their parent is selected. Child features may also be organized into Alternative (△) or Or (▲) groups. For Alternative groups, precisely one feature must be selected on selection of the parent feature, and at least one feature for Or groups. In our example, $B$, $C$, and $F$ are child features of $A$, with $B$ and $F$ being optional and $C$ being mandatory. $D$ and $E$ take part in an Alternative group and have $C$ as their parent feature. Additional constraints may be specified as Boolean expressions called *cross-tree constraints*. In our example, a selection of the feature $D$ implies a selection of the feature $B$.

In order to reason about the variability modeled in a feature model, the latter is typically transformed into a propositional formula in conjunctive normal form (CNF) [7, 8, 21]. Afterwards, SAT and #SAT solvers may be used to facilitate a wide range of analyses [8, 55]. The potentially large number of respective solver queries may also be replaced with knowledge compilation [54], where a formula is compiled into a data structure that allows for polynomial-time querying. In feature-model analysis, both d-DNNFs [16, 56] and binary decision diagrams [13, 22, 29] are typical targets of knowledge compilation.

### 2.2 Binary Decision Diagrams

Binary decision diagrams (BDD) are rooted, directed, and acyclic graphs with two terminal nodes, $\boxed{0}$ and $\boxed{1}$ [13]. All non-terminal nodes are associated with a variable $x$ and have two outgoing edges, $uu_-$ (the *low* edge) and $uu_+$ (the *high* edge), which encode the assignment of false (0) and true (1) to $x$, respectively. Typically, the low edge is drawn dashed. For a BDD $B$, we refer to its set of nodes by $V(B)$ and to its set of edges by $E(B)$, as usual in graph theory [18].

In addition and as usual [13, 19, 29], we always assume BDDs to be ordered and reduced [13]. We denote $BDD(F(M), \pi)$ for a BDD with variable order $\pi$ that was compiled from a Boolean formula $F$ and encodes the variability of a feature model $M$. The BDD in Figure 1b is ordered with variable ordering $\pi = (B, E, D, A, C, F)$ and reduced. Therefore, no nodes for the variable $F$ exist, as no variable assignment to $F$ has any impact on the validity of the configuration (recall that $F$ is optional).

For reduced and ordered BDDs $B$, every valid configuration of the feature model $M$ correlates uniquely to a path from the root node to $\boxed{1}$ in $B$ (called *1-path*). Note that the inverse is not a true,

as due to free variables and the BDD being reduced, such a 1-path may only induce a partial configuration that allows for undecided features to be arbitrarily selected or deselected. This can be observed in Figure 1c, where both the configurations $\{\overline{B}, E, \overline{D}, A, C, \overline{F}\}$ and $\{\overline{B}, E, \overline{D}, A, C, F\}$ are induced by the red 1-path.[1] Likewise, the blue 1-path also induces two configurations based on the partial configuration $\{B, \overline{E}, D, A, C\}$ and the freely selectable feature $F$.

## 2.3 T-Wise Sampling

Without additional constraints, two features $A$ and $B$ give way to four interactions, namely $\{A, B\}$, $\{A, \overline{B}\}$, $\{\overline{A}, B\}$, and $\{\overline{A}, \overline{B}\}$. T-wise sampling generates a set of valid configurations (a *sample*) for which it holds that each interaction between $t$ features is contained in at least one configuration. As, especially for $t > 2$, covering all $t$-wise interactions may already require a sample of impractical size, it is a common strategy to sacrifice some coverage in order for a reduced sample size [26, 28].

More formal, let $\mathbb{I}_t^*(M)$ denote the set of all valid interactions between $t$ features in a feature model $M$. For a sample $S$, let $\mathbb{I}_t(M, S)$ denote the set of $t$-wise interactions covered by configurations in $S$ and let

$$\kappa_t = \kappa_t(M, S) = \frac{|\mathbb{I}_t(M, S)|}{|\mathbb{I}_t^*(M)|}$$

denote the $t$-wise coverage $\kappa_t$ achieved by the sample $S$. For illustration, consider again the four 2-wise interactions from above and a sample $S = \{\{A, B\}, \{A, \overline{B}\}\}$. This sample covers two of the four interactions, hence

$$\kappa_2\big(\{\{A, B\}, \{A, \overline{B}\}, \{\overline{A}, B\}, \{\overline{A}, \overline{B}\}\}, \{\{A, B\}, \{A, \overline{B}\}\}\big) = \frac{2}{4} = 50\,\%$$

# 3 MOTIVATION AND STATE OF THE ART

In the following, we give an overview on related work on $t$-wise sampling and outline the challenges this work addresses.

Testing a configurable system requires a test suite and a set of configurations. In testing, the test suite is then executed for each of these configurations [25]. Different approaches exist to generate these configurations, such as manual configuration by experts [25, 45], usage-based feature selection [17, 25], feature-selection strategies [25, 41], biased random configuration [2, 25, 41], uniform sampling [25, 27, 43], and $t$-wise sampling [25, 28, 36, 41, 46]. This multitude of approaches resulted in a number of samplers, especially for uniform sampling [3, 14, 20, 27, 43] and $t$-wise sampling [4, 6, 36, 40, 42]. This, naturally, prompts the questions "which sampler to use?" and, for randomized or portfolio samplers,[2] "which sample to use?".

For uniform sampling, there are straightforward answers to these questions. As the sample size can be chosen freely, the only qualitative criteria remaining is the uniformity of the sample [27, 47]. However, for $t$-wise sampling, both the size of a sample and its coverage need to be considered. As discussed above, a high $t$-wise coverage is meaningless, when it forces an impractically large sample size [25]. Vice versa, a tiny sample size is of little value when too few interactions are covered. Combined, a small sample with high coverage remains impractical, when its computation is too time- or resource intensive. And, as we will discuss in the following, the value for $t$ must be considered as well [25, 41].

With regards to choosing a value for $t$, both Medeiros et al. [41] and Halin et al. [25] agree that $t = 3$ will typically suffice in practice. Heß et al., however, argue that for many real-world feature models, $t > 2$ is unrealistic both due to the resulting sample size as well as the time and resource requirements [28]. Recent work by Krupke et al. supports this plea by lower-bounding the size of 2-wise samples for a number of real-world feature models [37]. Even comparably small systems, such as Fiasco and uclibc require hundreds of configurations, matching similar previous observations [45]. financialservices01, a financial product line, which is often used as a benchmark [6, 35, 36, 46], even requires thousands of configurations [37].

Consequentially, we limit ourselves to $t = 2$ in this work. Nevertheless, we follow the commonly accepted argument that any statements for $t = 2$ also offer insights for $t > 2$ [6, 40, 44]. This leaves us with sampling time, sample size, and 2-wise coverage as currently used metrics for sample quality [12, 28, 36, 42, 46]. Both sampling time and sample size can easily be bound and measured [46], but measuring the coverage of a sample is, in general, an expensive operation in both time and memory [40].

The cost of coverage computations not only impacts reasoning about sample quality, but also the generation of samples. For instance, both Baital [6] and LS-Sampling [40] incrementally select configurations based on the number of additionally covered 2-wise interactions, again due to $t > 2$ entailing unreasonable costs. For post-processing approaches, such as greedily selecting configurations that cover the largest number of uncovered interactions, the cost of repetitive coverage computations is the greatest inhibitor [28]. Finally, for large systems, such as automotive02v4, computing the coverage of samples, or just counting the number of additional interactions covered by a configuration, is practically infeasible. Hence, automotive02v4 is typically excluded from benchmarks [6, 28, 40].

We conclude from the present state of the art that there is demand for a metric that can be efficiently computed and that takes both sample size and coverage into account. Besides judging the quality of samples, we envision multiple other applications, such as approximate sampling (i.e., not aiming for 100 % coverage), portfolio-based sampling,[3] and iterative postprocessing strategies that reduce sample size or increase coverage [28]. In the following, we introduce the pick-me (**pm**) metric, a BDD-based metric with the desired properties that can be efficiently computed for all models to which BDDs scale, including automotive02v4.

# 4 2-WISE INTERACTIONS ON BDDS

Recent advances in the scalability of BDD compilation [19, 29] make it feasible to compile BDDs for a wide range of feature models, including hard benchmarks for $t$-wise sampling, such as financialservices01 or automotive02v4. We harness this availability of BDDs in two ways. For one, we present an algorithm for exactly counting the number of 2-wise interactions on BDDs as well as generating all

[1] $X$ denotes the selection and $\overline{X}$ the deselection of a feature $X$.

[2] Like portfolio solvers, portfolio samplers run multiple samplers (multiple times) to enhance robustness and performance in practice

[3] I.e., running potentially multiple samplers potentially many times.

such interactions (cf. Section 4.1). Second, we present a novel metric that relates sample quality to edge usage in BDDs (cf. Section 4.2).

## 4.1 Counting Interactions on BDDs

As mentioned before (cf. Section 2), a valid configuration for a feature model $M$ uniquely translates to a path from the root node to the 1-terminal in $BDD(M, \pi)$. For two BDD levels $i, j$ with $i < j$, it directly follows from the BDD being ordered that a variable $v_{\pi(i)}$ is either decided or free on all paths leading into a node associated with $v_{\pi(j)}$ (cf. Figure 1b). When considering all nodes associated with $v_{\pi(j)}$, one therefore knows which interactions between the literals of $v_{\pi(0)}, \ldots, v_{\pi(j)}$ are possible, as they are present on at least one path from the root node to one of the nodes associated with $v_{\pi(j)}$.

Algorithm 1 exploits these insights to count the number of 2-wise interactions for a feature model $M$ encoded by a BDD $BDD(M, \pi) = B$. First, it is directly known in $B$ which variables are unconstrained, due to $B$ being reduced (cf. Section 2). Hence, the number of interactions between the $2n$ literals of these $n$ free variables can be directly computed by subtracting the $n$ interactions involving literals of the same variable from the total number of interactions between the $2n$ literals (cf. Line 1):

$$\frac{\binom{2n}{2}}{2} - n = n \cdot (2n-1) - n = n \cdot (2n-2) = 2n \cdot (n-1)$$

Afterwards, two mappings are defined. $\mathcal{L}$ maps a BDD node $u$ to the set of literals available at $u$, meaning the union of partial configurations whose paths lead to $u$ (cf. Line 2). The second mapping $\mathcal{I}$, maps a literal $l$ to the set of all literals that interact with $l$ (cf. Line 3). By construction, all literals in $\mathcal{I}(l)$ belong to variables that are decided ahead of $u$'s level in the variable order.

The algorithm traverses a BDD breadth-first, level by level, starting with the first level that only contains the root node. For each node $u$, all literals available at $u$ are propagated to its child nodes $u_-$ and $u_+$, together with the respective assignment for the variable $v$ associated with $u$.

As it is often the case with reduced and ordered BDDs, the edges $uu_+$ and $uu_-$ may "jump" over levels associated with variables that are locally free with respect to all partial configurations running through $u$. The algorithm accounts for this in two ways. For one, the subroutine Algorithm 2 is called to collect the literals of these free variables. Second and instead of many set operations in Algorithm 2, we need to handle these additional literals when counting the interactions (cf. Lines 21f).

Once all nodes associated with a variable $v$ have been visited, the sets $\mathcal{I}(v)$ and $\mathcal{I}(\overline{v})$ are known, namely all literals from variables associated with previous levels that form interactions with $v$ and $\overline{v}$. Note that additional interactions of $v$ and $\overline{v}$ are, by construction, only possible with free variables and variables associated with nodes in lower levels. If a variable $v$ is neither dead nor core, both its literals will interact with all literals of the free variables, hence we add twice the cardinality of $V_{free}$ in this case.

In the same manner, all levels in the BDD are traversed, with the number of interactions being summed up and returned at the end. Note that $\mathcal{L}(u)$ can be dropped after $u$ has been visited, as can $\mathcal{I}(v)$ and $\mathcal{I}(\overline{v})$ once they have been counted. Finally, note that the algorithm can be easily augmented to not account for interactions with core or dead variables [11], or to yield the interactions instead of merely counting them.

Our algorithm fills the scalability gap for models, such as automotive02v4, for which it is easy to compile BDDs [29] but time and memory intensive to count the number of 2-wise interactions by other means. However, it scales linearly with the number of nodes in a BDD. Therefore, classical approaches to interaction counting will likely be superior for models that require large BDDs, for instance the CDL models [9]. In contrast, the metric we present in the following scales linearly with the number of levels or the sample size and can therefore be efficiently computed on even the largest BDDs.

**Algorithm 1:** Counting 2-wise Interaction on BDDs

**Input:** $BDD(M, \pi)$
**Output:** number $N$ of 2-wise interactions in $M$

1 $N \leftarrow 2\,|V_{free}| \cdot (|V_{free}| - 1)$
2 $\mathcal{L}(u) \leftarrow \emptyset, \forall\, u \in Nodes(BDD(M, \pi))$
3 $\mathcal{I}(l) \leftarrow \emptyset, \forall$ literals of $v \in V(M)$
4 **foreach** $i \in \{1, \ldots, |\mathcal{F}(M)|\}$ **do**
5  **if** $\pi(i) \in V_{free}$ **then**
6  **continue**
7  **endif**
8  **foreach** *node u associated with* $v_{\pi(i)}$ **do**
9  $L \leftarrow \mathcal{L}(u)$
10  **if** $u_+ \neq \boxed{\mathbf{0}}$ **then**
11  $\mathcal{I}(\mathtt{var}(u_+)) \leftarrow \mathcal{I}(u_+) \cup L$
12  $\mathcal{L}(u_+) \leftarrow \mathcal{L}(u_+) \cup \{v_{\pi(i)}\} \cup L$
13  $\mathcal{I},\ \mathcal{L} \leftarrow \mathtt{HandleFree}(u,\ u_+,\ \mathcal{I},\ \mathcal{L})$
14  **endif**
15  **if** $u_- \neq \boxed{\mathbf{0}}$ **then**
16  $\mathcal{I}(\mathtt{var}(u_-)) \leftarrow \mathcal{I}(u_-) \cup L$
17  $\mathcal{L}(u_-) \leftarrow \mathcal{L}(u_-) \cup \{\overline{v_{\pi(i)}}\} \cup L$
18  $\mathcal{I},\ \mathcal{L} \leftarrow \mathtt{HandleFree}(u,\ u_-,\ \mathcal{I},\ \mathcal{L})$
19  **endif**
20  **end**
21  $n \leftarrow |\{l \in \mathcal{I}(v_{\pi(i)}) \mid \pi^{-1}(|l|) < i\}|$
22  $m \leftarrow |\{l \in \mathcal{I}(\overline{v_{\pi(i)}}) \mid \pi^{-1}(|l|) < i\}|$
23  **if** $v_{\pi(i)}$ *is not dead* **then**
24  $n \leftarrow n + 2\,|V_{free}|$
25  **endif**
26  **if** $v_{\pi(i)}$ *is not core* **then**
27  $m \leftarrow m + 2\,|V_{free}|$
28  **endif**
29  $N \leftarrow N + n + m$
30 **end**
31 **return** $N$

## 4.2 Sample Quality Metric

We now present our novel **pm** metric to judge the quality of a (t-wise) sample. Its goal is to rank samples for the same system, with the

**Algorithm 2:** Handling free variables on skipped nodes

**Input:** BDD node $u$ with child node $w$, $\mathcal{L}$, $\mathcal{I}$
**Output:** Updated interactions in $\mathcal{I}$, updated ingoing literals in $\mathcal{L}$

```
foreach j ∈ {π⁻¹(var(u)) + 1, ..., π⁻¹(var(w)) − 1} do
    I(v̄_π(j)) ← I(v̄_π(j)) ∪ L(w)
    I(v_π(j)) ← I(v_π(j)) ∪ L(w)
    L(w) ← L(w) ∪ {v_π(j), v̄_π(j)}
end
return I, L
```

"best" sample achieving the highest score. The core intuition behind the metric is the observation that two valid configurations, together, cover more interactions when they are more disjoint. This can be easily seen, as the number of interactions redundantly covered by both $A$ and $B$ grows with the size of their intersection.

In an ideal world, one would count the number of uniquely covered interactions per configuration and could then define a metric based on its average or similar statistics. However, such approaches scale quadratically with the number of variables in time or memory and are therefore expensive to compute. However, we can proxy this idea on BDDs, as described in the following.

As before, we exploit the fact that each valid configuration uniquely maps to a distinct 1-path in the BDD. Two configurations will only map to the same 1-path, when all literals in their difference correspond to free variables on the path. Vice versa, two distinct 1-paths in the BDD will differ in at least one variable assignment and, therefore, will differ in at least one edge (cf. Figure 1c).

Putting it all together, we hypothesize that high-quality samples contain many disjoint configurations,[4] in order to cover more interactions with less configurations. Disjoint configurations will likely map to different 1-paths in the BDD and high-quality samples will, therefore, cover more BDD edges with fewer configurations. As a consequence, individual BDD edges will be used less frequently.

Figure 2 depicts the distribution of edge usages of a YASA sample for the financialservices01 model in a BDD $B$. The x-axis represents the edge usages of the samples, i.e., the number of times an edge is included in a 1-path. The bars and, respectively, the y-axis account for the relative number of such edges. For example, a bar at position $x$ of height $h$, denotes that $h$ of all edges are included in $x$ 1-paths.

Recall our argument from before that high-quality samples will visit more BDD edges in total but individual edges less frequently. Following this argument, we would expect that a high-quality sample predominantly has high bars in the lower left part of the plot. When summing up the frequencies of edges with respective usages, the resulting sum should therefore climb steeper for samples of higher quality than for others.

In order to get a representative slope for the sum of frequencies, we choose the average slope from start to the mean value of the cumulative sum of frequencies. Let $\texttt{freq}(n)$ denote the frequency of edges visited by $n$ 1-paths and $\Sigma\texttt{freq}(n)$ the cumulative sum of frequencies visited by 1 to $n$ 1-paths. Then, $\overline{\Sigma\texttt{freq}}$ denotes the mean of this cumulative sum and we choose $p$ as the first $x$-coordinate at which this mean is surpassed, i.e.,:

$$\Sigma\texttt{freq}(p-1) < \overline{\Sigma\texttt{freq}} \leq \Sigma\texttt{freq}(p)$$

Finally, this allows us to the define the aforementioned slope $\Delta$ as:

$$\Delta = \frac{\overline{\Sigma\texttt{freq}}}{p}$$

With that, we can now define the *"pick me"*-metric (**pm**) of a sample $S$ with respect to a BDD $B$ as:

$$\textbf{pm} = \frac{|E(B,S)|}{|S|} \cdot \Delta$$

In summary, **pm** employs a BDD to compute a numerical score for a given sample $S$ and a BDD $B$. In essence, it combines the ratio between edge usage and sample size together with the slope of the sum of edge-usage frequencies. Following our hypothesis, we claim that higher-quality samples will attain higher **pm** scores, by construction. Namely, reducing the sample size, increasing the number of covered edges, or increasing the slope of the frequencies' sum will all increase the **pm** score, respectively. Again, Note that **pm** is only suited to compare samples for the same feature model on the same BDD and that its absolute value is meaningless.

In the following evaluation, we investigate the veracity of our intuition and demonstrate the utility of **pm** for judging the quality of samples.

## 5 EVALUATION

In this evaluation, we assess the capability of the **pm** metric to judge the quality of 2-wise samples. Furthermore, we demonstrate the upsides of our BDD-based interaction-counting algorithm, i.e., that it scales to the automotive02v4 model. To this end, we investigate the following research questions:

**RQ1**: How efficiently can **pm** be computed?

**RQ2**: Is **pm** capable to distinguish between samples of high and low quality?

**RQ3**: Is **pm** sensitive to changes in sample quality?

**RQ4**: How does BDD-based interaction counting compare to the state of the art?

### 5.1 Preliminaries

*Environment.* The evaluation was executed within a Docker container on a laptop with a Ryzen 5 8645H CPU and 16 GB of RAM, using the `python:3.13` image. All preliminary steps were executed in parallel using 8 threads. The experiment steps were executed in a single thread.

*Subject Systems.* We use a variety of well-known real-world feature models from the collection of benchmark instances by Sundermann et al. [52]. However, as our evaluation requires BDDs, we only used models for which BDDs can be compiled in reasonable time [19, 29]. Nevertheless, due to recent advances in BDD compilation, we were able to include a number of challenging feature models [34, 46], from different domains. automotive01 and automotive02v4 model systems in the automotive industry [34] and financialservices01 models a financial product line [23]. am31_sim,

[4] Note that distance-based sampling is based on a similar hypothesis [26].

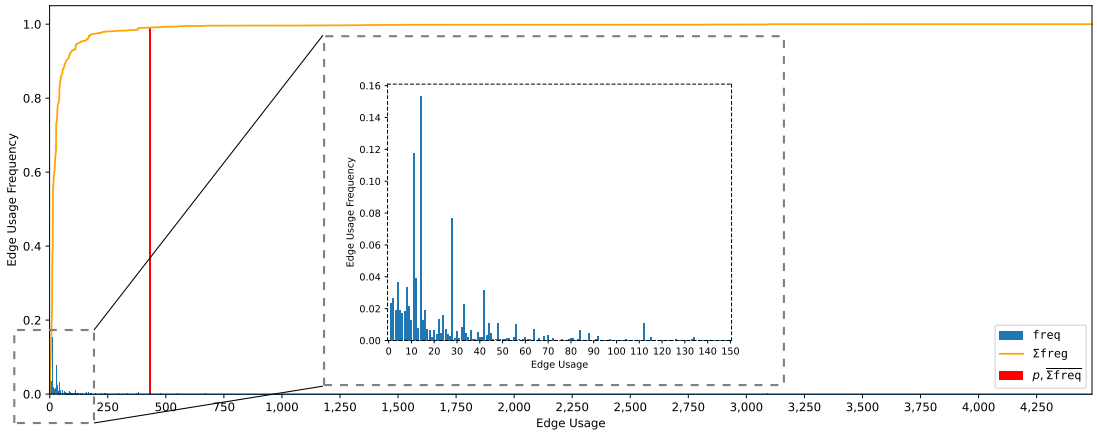



**Figure 2: Visualization of Δ for a YASA sample of financialservices01**

**Table 1: Statistics on the subject systems in our evaluation and their compilation into BDDs**

| Models | # Variables (raw) | | # Clauses (raw) | | Model Count | Pre Time (s) | BDD Time (s) | BDD Size |
|---|---|---|---|---|---|---|---|---|
| jhipster | 36 | (45) | 70 | (104) | 2.63e+4 | 0.00 | 0.01 | 152 |
| berkeleydb | 60 | (76) | 100 | (140) | 4.08e+9 | 0.00 | 0.01 | 509 |
| toybox | 78 | (544) | 92 | (1,020) | 1.45e+17 | 0.00 | 0.01 | 96 |
| fiasco | 209 | (1,638) | 999 | (5,228) | 3.58e+14 | 0.03 | 0.06 | 3,342 |
| financialservices01 | 645 | (771) | 6,736 | (7,238) | 9.75e+13 | 0.41 | 9.25 | 69,390 |
| embtoolkit | 588 | (1,179) | 3,341 | (5,414) | 5.13e+96 | 0.06 | 0.06 | 2,173 |
| am31_sim | 589 | (1,178) | 1,022 | (2,344) | 2.63e+118 | 0.16 | 14.40 | 2,818,696 |
| p2106 | 620 | (1,262) | 1,077 | (2,528) | 3.03e+124 | 0.11 | 14.61 | 3,426,118 |
| ecos-icse11 | 629 | (1,244) | 1,159 | (3,146) | 4.97e+125 | 0.15 | 15.38 | 3,281,702 |
| ea2468 | 652 | (1,408) | 1,142 | (2,808) | 4.81e+130 | 0.15 | 23.65 | 4,977,209 |
| busybox_1.18.0 | 792 | (854) | 542 | (1,163) | 2.06e+201 | 0.05 | 0.09 | 16,600 |
| automotive01 | 1,672 | (2,513) | 6,573 | (10,300) | 5.28e+210 | 0.78 | 9.37 | 602,093 |
| automotive02_v4 | 16,494 | (18,616) | 330,637 | (350,119) | 1.78e+1534 | 26.57 | 22.15 | 227,698 |

ea2468, p2106, and ecos-icse11 are CDL models [9, 38]. The remaining models were extracted from the kconfig modeling language [9, 39, 52].

In total we include 13 feature models, ranging from 45 to 18,616 variables,[5] from 104 to 350,119 clauses, and from 2,289 to over half a billion 2-wise interactions. The models' characteristics before (raw) and after preprocessing as well as statistics on the BDD compilation are depicted in Table 1. As usual, the size of a BDD is given by its number of nodes.

*Tools.* We use the well-known BDD library CuDD [51] for BDD compilation, which we interface with from Python using a `ctypes`[6] interface. For comparison and to validate the interaction counts and coverages, we use the FeatureIDE descendant FeatJar.[7] FeatJar also contains the current reference implementation of the t-wise sampler YASA [36]. Finally, we use the well-known model counter and uniform sampler Spur [3] to validate the correctness of all preprocessing steps. We also use Spur to generate known bad 2-wise samples [28, 42].

## 5.2 Experiment Setup

We conducted two experiments to answer the research questions from above. As both **`pm`** and the interaction-counting algorithm are BDD-based, we first compiled BDDs for the models in our evaluation, in a preliminary step that also includes equivalence-preserving preprocessing.

*Preprocessing.* We preprocessed the models' DIMACS files with atomic-set elimination [31, 50]. Atomic sets are sets of variables that must assume the same assignment in every valid configuration. These sets may be condensed into a single variable, while preserving equivalence. By undoing this step, samples for the original model

[5] As all CNFs where obtained without use of the Tseitin transformation [52], we use features and variables synonymously.
[6] https://docs.python.org/3/library/ctypes.html
[7] https://github.com/FeatureIDE/FeatJAR

can directly be derived from samples for the preprocessed model. Atomic-set elimination typically drastically reduces the numbers of clauses and variables [31] and therefore speeds up every part of this evaluation, without affecting its outcome.

*BDD Compilation.* We follow the insights of recent works on compiling feature models into BDDs, using a mix of the approaches of Dubslaff et al. and Heß et al. [19, 29]. In particular, we detect xor groups and exploit them for efficient variable ordering, when existing [29]. For the remaining models, we use a deterministic variant of the well-known FORCE heuristic [5] for variable ordering. Afterwards, we compile BDDs using window permutation [24, 32] for dynamic variable ordering. Table 1 contains statistics on the BDD compilation for the models in our evaluation.

*Experiment 1.* Our first experiment is concerned with evaluating our novel **pm** metric. First, we compute 2-wise samples with 100 % coverage using YASA [36] as well as uniform samples with 1024 configuration using Spur. As it is well known that uniform samples achieve poor $t$-wise coverage [28, 42], we would expect **pm** to be able to clearly distinguish between these samples.

Second, we measure the sensitivity of **pm** regarding changes to the size and coverage of a sample. As outlined in Section 4, we designed **pm** to yield a lower score for samples with same coverage but larger size or samples with same size but lower coverage. As it is infeasible to generate such samples directly, we resort to mutating the YASA samples. For increasing the sample size without affecting coverage, we duplicate a percentage of configurations and append them to the sample. For decreasing coverage, while maintaining sample size, we replace a percentage configurations by duplicates of non-replaced configurations.

*Experiment 2.* Decoupled from the first experiment, we compare the efficiency of counting the number of 2-wise interactions with both our BDD-based algorithm and FeatJar.

## 5.3 Results

We now discuss the results obtained from our experiments. First and foremost, both the preprocessing [31] and the compilation into BDDs performed as reported in previous works (cf. Table 1) [19, 29]. Likewise, Spur and YASA also scaled as expected [27, 28, 36], with YASA not scaling to the automotive02v4 feature model [36, 46].

Based on the computed BDDs and samples, we may now investigate the performance of computing the **pm** metric. Among others, Table 2 lists the computation times for **pm** and classical coverage computation. While both lie within measurement uncertainty for the small models, the differences in computation cost become very pronounced for the larger models with larger sample sizes.

In particular, one can make three observations. First, beyond some overhead, **pm** is linear with respect to the sample size. This can be observed when comparing the costs for computing **pm** for Spur and YASA samples (recall that all Spur samples are of size 1024). Second, **pm** appears to be not impacted by BDD size. In fact, computing **pm** is more expensive for financialservices01 (e.g., 0.38 s for the YASA sample) than for ea2468 (e.g., 0.23 s for the YASA sample), despite the latter requiring almost two orders of magnitude more BDD nodes. Third, for all but the smallest three models, **pm** is commonly at least an order of magnitude faster to compute in comparison to classical coverage, while also not requiring nearly as much memory.

Of course, the efficiency advantages of **pm** are only meaningful when it is capable of judging sample quality. Figure 3 depicts the scores achieved by the dedicated 2-wise sample computed with YASA and the uniform sample computed with Spur. Recall that we expect the scores of the YASA samples to be higher. Indeed, this is the case for all models but embtoolkit. The deviation for the latter is readily explained by the enormous size of its YASA sample, requiring 2,548 configurations to achieve 100 % coverage. Thereby it is more than double the size of the Spur sample with 1,024 configurations.

For two other systems, namely automotive01 and financialservices01, the score difference is comparably small, albeit YASA achieving the higher score both times. In the case of automotive01, this is due to the Spur sample already achieving 93.5 % 2-wise coverage with 1,024 configurations, while the YASA sample contains 1,628 configurations to achieve 100 %. financialservices01 on the other hand, is infamous for requiring many configurations (here 4,488) to achieve 100 % 2-wise coverage [6, 28, 35–37, 42]. Due to its structural properties, uniform sampling achieves especially poor coverage for this model [28, 42], explaining why a four times difference in size does not suffice for the Spur sample to surpass the YASA sample in the **pm** metric.

The boxplots in Figure 3 clearly indicate that the **pm** metric is capable of distinguishing between known good and known bad samples. Indeed, on average, YASA and Spur are distinguished by over an order of magnitude on average.

With **pm** being able to distinguish between samples with drastic differences in sample quality, we continue with evaluating its sensitivity towards smaller changes to a sample. First, we increased the sample size by appending duplicate configurations. Figure 4 shows the development of **pm** for all systems and size increases of 5 % to 50 %. One can clearly see that the **pm** scores develop as expected, getting lower with each increase in size. This is both evident in the line plots for the respective systems but also in the development of the box plots.

For measuring the sensitivity of **pm** with respect to the coverage achieved within a sample, we replace configurations by duplicates of other configurations, in order to maintain sample size but lowering the coverage. Figure 5 contains line and box plots depicting the development of **pm** scores while replacing 5 % – 50 % of configurations with duplicates. While the score reduction is not as distinct as before for most models (cf. Figure 4), it nevertheless decreases on average. Note that this and even intermediate score increases are to be expected for a number of reasons, we discuss in the following.

In fact and as demonstrated by other works, few configurations often suffice to still cover more than 90 % of interactions [28]. As a direct consequence, samples with thousands of configurations will likely cover many interactions with multiple configurations. The other way round, many configurations will only cover few unique interactions. Put together, replacing a few configurations with duplicates is unlikely to drastically decrease coverage, especially for large samples, such as for financialservices01, with 4,488 configurations. On the other hand, for systems that require far fewer configurations for full coverage, such as the CDL models (116 configurations on average), one would expect significant changes

**Table 2: Computation times for interaction counting, 2-wise sampling, and metric computation**

| Model $M$ | $\lvert\mathbb{I}_2^*(M)\rvert$ | Counting Interactions (Time, s) | | | YASA Sampling | | Time (s) for Metric Computations | | |
|---|---|---|---|---|---|---|---|---|---|
| | | YASA | BDD | BDD (♠) | Time | Size | **pm** (Yasa) | **pm** (Spur) | cov (Yasa) |
| jhipster | 2,289 | 0.37 | 0.01 | 0.00 | 0.45 | 39 | 0.00 | 0.00 | 0.02 |
| berkeleydb | 6,490 | 0.41 | 0.03 | 0.01 | 0.49 | 26 | 0.02 | 0.01 | 0.03 |
| toybox | 11,603 | 0.37 | 0.16 | 0.15 | 0.35 | 16 | 0.02 | 0.00 | 0.00 |
| fiasco | 75,405 | 0.73 | 0.08 | 0.02 | 0.86 | 167 | 0.00 | 0.02 | 0.25 |
| financialservices01 | 655,186 | 5.84 | 9.91 | 0.66 | 11.24 | 4,488 | 0.38 | 0.08 | 2.57 |
| embtoolkit | 684,332 | 1.61 | 0.49 | 0.43 | 6.73 | 2,086 | 0.14 | 0.06 | 4.79 |
| am31_sim | 684,660 | 0.77 | 89.52 | 75.13 | 1.29 | 109 | 0.13 | 0.08 | 0.25 |
| p2106 | 759,235 | 0.79 | 100.15 | 85.55 | 0.95 | 114 | 0.15 | 0.09 | 0.29 |
| ecos-icse11 | 782,646 | 0.83 | 114.99 | 99.60 | 1.60 | 123 | 0.17 | 0.09 | 2.58 |
| ea2468 | 839,962 | 0.84 | 158.74 | 135.10 | 1.50 | 116 | 0.23 | 0.09 | 2.76 |
| busybox_1.18.0 | 1,249,184 | 0.73 | 0.47 | 0.38 | 1.60 | 70 | 0.00 | 0.06 | 0.31 |
| automotive01 | 5,563,298 | 4.09 | 32.58 | 23.20 | 42.93 | 1,628 | 0.52 | 0.26 | 32.04 |
| automotive02_v4 | 543,672,009 | ♦ | 167.13 | 144.98 | ♦ | | | 2.29 | ♦ |

♦ = timeout, = best, = best when ♠, ♠ = without BDD compilation

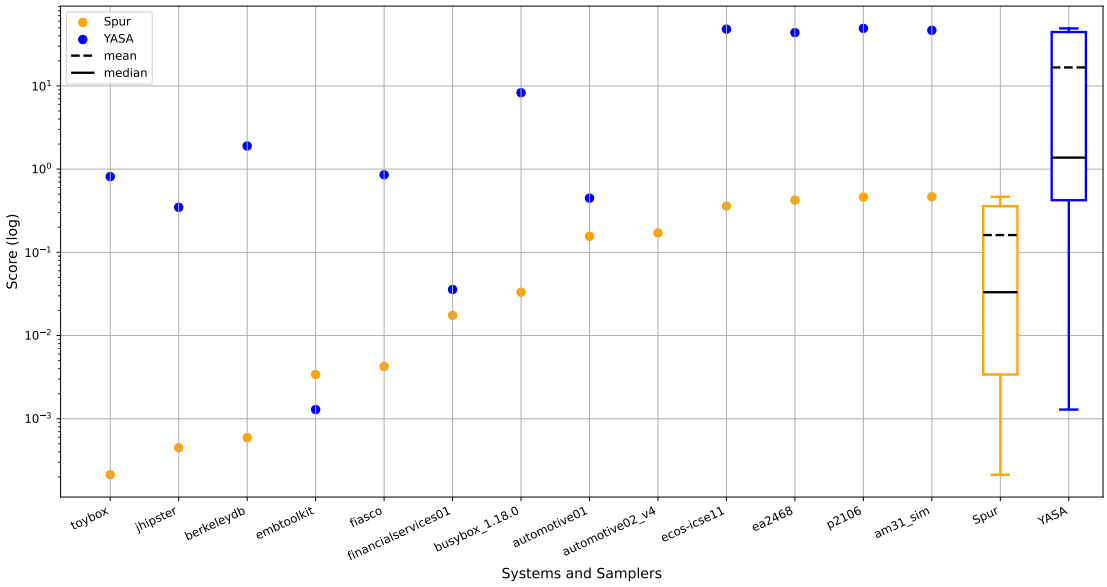



**Figure 3: pm scores achieved by Spur and YASA for the subject systems (higher is better)**

in coverage and therefore **pm**, due to mutation. Figure 5 depicts that this is the case.

*Significance of* **pm** *Sensitivity.* While Figure 4 and Figure 5 provide some visual evidence that the **pm** metric is sensitive to changes in both sample size and coverage, they require additional discussion to be read correctly. To corroborate our claims, we also computed the correlations between mutations and score decreases as well as their significances.

Table 3 lists the Pearson correlation coefficient and its significance for all systems. For all systems and on average, increasing the sample size by mutation significantly correlated with a decrease in **pm** score (strong negative correlation, $\rho \approx -0.98$ and $p \approx 6.56 \cdot 10^{-8} \ll 0.05$.

We achieved similar results for the coverage mutation, also listed in Table 3, with $\rho \approx -0.97$ and $p \approx 2.16 \cdot 10^{-5} \ll 0.05$. This means that increased mutation and therefore a reduction in coverage significantly correlates with a decrease in **pm** score. The two outliers, embtoolkit and financialservices01 with $p > 10^{-6}$, can be explained

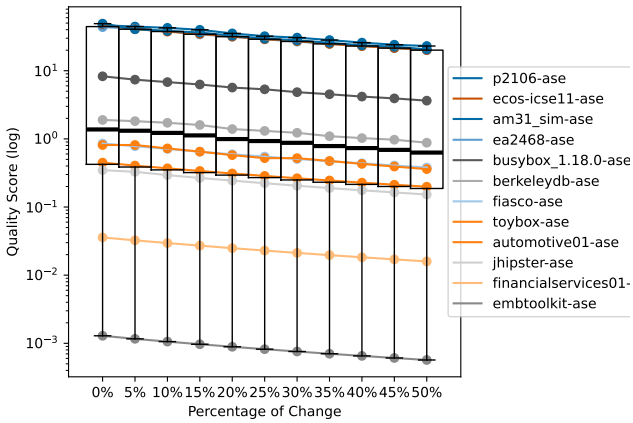



**Figure 4: Score Development During Size Increase**

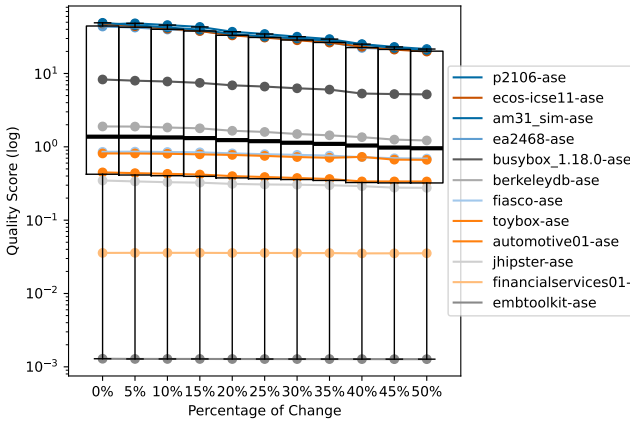



**Figure 5: Score Development During Coverage Reduction**

with a similar argument as before. Due to their large sample sizes with respect to the number of interactions, many configurations will only cover few interactions uniquely. Therefore, it is feasible that the score stagnates or even increases, when $\Delta$ proportionally increases more than $|E(B, S)|$ decreases. Nevertheless, the correlation remains significant in both cases.

*Efficiency of Counting Interactions.* Consider again Table 2, which also lists the times required for counting the number of 2-wise interactions with YASA and our novel BDD-based approach. For the sake of completeness, we listed both the times with and without BDD compilation. In any case, using a BDD for counting interactions outperforms FeatJar/YASA for all models but the CDL models, financialservices01, and automotive01. When we disregard the compilation time for the BDD, our approach is also faster for financialservices01.

Furthermore and as expected, we can see that the cost of our BDD-based approach is directly impacted by the size of the BDD. Finally, we note that we are, to the best of our knowledge, the first to efficiently compute the number of 2-wise interactions for automotive02v4 [28, 36].

## 5.4 Discussion

Our evaluation clearly shows that computing the **pm** metric can be done efficiently, especially in comparison to related tasks, such as sample generation, interaction counting, and coverage computation (cf. Table 2). This efficiency is also upheld for hard benchmark instances, such as automotive02v4. Therefore, we may answer **RQ1** "How efficiently can **pm** be computed?" favorably and conclude that computing **pm** can be done efficiently.

With regards to **RQ2** "Is **pm** capable to distinguish between samples of high and low quality?", we conclude that **pm** is indeed capable. Our evaluation shows that **pm** allows for both judging the sample quality of samplers as well as the quality of individual samples (cf. Figure 3). Not only does **pm** distinguish the quality clearly, the analysis of cases with small differences, provided valuable insights on the samples and systems.

**RQ3** asked whether **pm** is also capable of detecting small changes to samples. Our analysis of the correlations and its significances shows that the **pm** score significantly and strong negatively correlates to size increases and coverage decreases and, therefore, strong positively correlates to size decreases and coverage increases. Our significances are orders of magnitude better than the typical threshold of $p < 0.05$. Our only outlier, embtoolkit, does not correlate strongly ($|\rho| = 0.89 < 0.9$). However, we would argue that the **pm** metric is nevertheless conclusive here, there is simply not enough difference between the sample and its mutations for the metric to value.

Our results for **RQ4** "How does BDD-based interaction counting compare to the state of the art?" are also noteworthy, but less exciting. The results show that BDD-based interaction counting outperforms the SAT-solving-based counting in YASA, as long as the BDDs remain small enough, with the threshold occurring at around 250,000 BDD nodes. (cf. Table 1 and Table 2). We want to highlight that the computation times include the times for BDD compilation, which was unthinkable just a few years ago [30]. To the best of our knowledge, our BDD-based interaction counting algorithm is the first that scales to automotive02v4 in minutes. Thereby, we close an important gap [6, 35, 36, 46] and pave the way for extending future evaluations to more complex systems.

## 5.5 Threats to Validity

*Internal Validity.* The outcome of our experiments could be disturbed by a number of factors. For one, any of processing steps could be faulty, starting from model extraction, translating the model into CNF, preprocessing the CNF, compiling a BDD from the CNF. Naturally, the same holds for the samplers we use and the algorithms we use to measure coverage or the number of interactions.

However, the models in our evaluation and their respective CNFs are well-known and have been used in many empirical works [19,

**Table 3: Correlation and Significance Between Sample Mutation and `pm` Score**

| | Size Reduction | | Coverage Reduction | |
|---|---|---|---|---|
| Models | Correlation ($\rho$) | Significance ($p$) | Correlation ($\rho$) | Significance ($p$) |
| am31_sim | -0.992 | 1.90e-09 | -0.987 | 1.89e-08 |
| automotive01 | -0.985 | 3.15e-08 | -0.990 | 6.69e-09 |
| berkeleydb | -0.992 | 1.64e-09 | -0.993 | 1.37e-09 |
| busybox_1.18.0 | -0.986 | 3.02e-08 | -0.992 | 1.72e-09 |
| ea2468 | -0.989 | 8.30e-09 | -0.993 | 1.02e-09 |
| ecos-icse11 | -0.977 | 2.65e-07 | -0.992 | 1.79e-09 |
| embtoolkit | -0.985 | 3.81e-08 | -0.938 | 1.99e-05 |
| fiasco | -0.986 | 2.52e-08 | -0.980 | 1.35e-07 |
| financialservices01 | -0.986 | 2.50e-08 | -0.890 | 2.38e-04 |
| jhipster | -0.987 | 2.04e-08 | -0.993 | 1.30e-09 |
| p2106 | -0.976 | 2.96e-07 | -0.994 | 8.01e-10 |
| toybox | -0.984 | 4.44e-08 | -0.970 | 7.64e-07 |

27–29, 34, 52, 53]. Nevertheless we verified the results of each processing steps, for instance by comparing the numbers of valid configurations to the literature and with off-the-shelf #SAT solvers. All tests were successful.

Furthermore, as the metric's value depends on the BDD, changes to the BDD could lead to different results. For one, we argue that the central property we exploit, the unique mapping of configurations to 1-paths, holds for all BDDs. We would have expected this behavior to occur for at least one BDD in our evaluation. Therefore, we are confident that this is not the case.

Finally, our randomized mutation of the samples could skew the outcome of the sensitivity measurements. However, we conducted ten decoupled mutations per sample, ranging from 5 % to 50 % of configurations, without encountering outliers. Intermediate deviations from the trend can be explained with the nature of the respective models and samples. Consequently, we are confident that the measured correlation and significance are valid.

*External Validity.* We can not argue that our findings will translate to all feature models. For one, we were, naturally, limited to feature models for which BDD compilation is feasible. While this, at present, includes many feature models from a variety of domains, this is a limiting factor of our approach. Nevertheless, the BDDs in our evaluation are very diverse and, for instance, ranged in size from hundreds of nodes to millions of nodes. Due to the high significance of our results, we would therefore expect that our findings translate to other feature models, for which BDDs may be compiled.

## 6 RELATED WORK

While we already discussed works that we built upon or that motivate our work, we now briefly discuss other related works. In particular, we discuss alternative approaches to sampling when testing configurable systems.

First and foremost, the selection of configurations by experts remains an important approach to selecting configurations for testing [25, 45]. To avoid bias [25], this prioritization of configurations can be based on statistical evidence, such as usage data [17] or the distance between configurations [26, 49]. Halin et al. demonstrated that these approaches can be competitive in terms of fault detection to $t$-wise samples of same size.

Beyond product-based analyses, family-based analyses simultaneously analyze all configurations of a configurable system by encoding variability directly into the analysis model [58]. Prominent approaches annotate source-code derivatives such as abstract syntax trees [33], control-flow graphs [57], or featured transition systems [15] with presence conditions. Rhein et al. showed that family-based approaches compare favorably on real-world systems against sampling-based approaches for a number of practical control-flow and data-flow analyzes [48]. On the downside, family-based analyzes are harder to adopt, to execute, and many faults may be found without the need fo exhaustive analysis [48].

## 7 CONCLUSION

To the best our knowledge, we are the first to apply BDDs to the context of combinatorial interaction analysis. We presented two contributions, an algorithm for counting the number of 2-wise interactions on BDDs and a novel metric for judging sample quality. Our evaluation shows that our interaction-counting algorithm closes the scalability gap for large but under-constrained models. Furthermore, we demonstrated that our metric is (1) fast to compute, (2) able to distinguish good from bad samples, and (3) sensitive to minor changes in sample size and coverage.

We conclude that this work constitutes a new hammer in the feature-model analysis toolbox. Our evaluation demonstrated that BDD-based interaction counting is practical alternative to SAT-solving-based counting, as long as the BDDs are smaller than 250,000 nodes. However, the major contribution of this work is the **`pm`** metric, which allows to almost instantaneously compare sample quality, when a BDD is available. In particular, **`pm`** appears to be independent of BDD size or the number of 2-wise interactions.

## REFERENCES

[1] Iago Abal, Jean Melo, Stefan Stănciulescu, Claus Brabrand, Márcio Ribeiro, and Andrzej Wąsowski. 2018. Variability Bugs in Highly Configurable Systems: A Qualitative Analysis. *Trans. on Software Engineering and Methodology (TOSEM)* 26, 3, Article 10 (2018), 10:1–10:34 pages.

[2] Mathieu Acher, Hugo Martin, Juliana Alves Pereira, Arnaud Blouin, Djamel Eddine Khelladi, and Jean-Marc Jézéquel. 2019. *Learning From Thousands of Build Failures of Linux Kernel Configurations.* Technical Report. Inria ; IRISA. 1–12 pages. https://inria.hal.science/hal-02147012

[3] Dimitris Achlioptas, Zayd S Hammoudeh, and Panos Theodoropoulos. 2018. Fast Sampling of Perfectly Uniform Satisfying Assignments. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT).* Springer, 135–147.

[4] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. 2016. IncLing: Efficient Product-line Testing Using Incremental Pairwise Sampling. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE).* ACM, 144–155.

[5] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. 2003. FORCE: A Fast and Easy-to-Implement Variable-Ordering Heuristic. In *Proc. ACM Great Lakes Symposium on VLSI (GLSVLSI).* ACM, 116–119.

[6] Eduard Baranov, Axel Legay, and Kuldeep S. Meel. 2020. Baital: An Adaptive Weighted Sampling Approach for Improved t-Wise Coverage. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE).* ACM, 1114–1126.

[7] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* Springer, 7–20.

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

[9] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Trans. on Software Engineering (TSE)* 39, 12 (2013), 1611–1640.

[10] Antonia Bertolino. 2007. Software Testing Research: Achievements, Challenges, Dreams. IEEE, 85–103. doi:10.1109/FOSE.2007.25

[11] Sabrina Böhm, Sebastian Krieter, Tobias Heß, Thomas Thüm, and Malte Lochau. 2024. Incremental Identification of T-Wise Feature Interactions. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS).* ACM, 27–36.

[12] Sabrina Böhm, Tim Schmidt, Sebastian Krieter, Tobias Pett, Thomas Thüm, and Malte Lochau. 2025. Coverage Metrics for T-Wise Feature Interactions. In *Proc. Int'l Conf. on Software Testing, Verification and Validation (ICST).* IEEE. To appear.

[13] Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Computers* C-35, 8 (1986), 677–691.

[14] Supratik Chakraborty, Daniel J Fremont, Kuldeep S Meel, Sanjit A Seshia, and Moshe Y Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In *Proc. Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).* Springer, 304–319.

[15] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. 2010. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *Proc. Int'l Conf. on Software Engineering (ICSE).* ACM, 335–344.

[16] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *J. Artificial Intelligence Research (JAIR)* 17, 1 (2002), 229–264.

[17] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Hamza Samih, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2017. Statistical Prioritization for Software Product Line Testing: An Experience Report. *Software and Systems Modeling (SoSyM)* 16, 1 (2017), 153–171. doi:10.1007/S10270-015-0479-8

[18] Reinhard Diestel. 2012. *Graph Theory, 4th Edition.* Graduate Texts in Mathematics, Vol. 173. Springer.

[19] Clemens Dubslaff, Nils Husung, and Nikolai Käfer. 2024. Configuring BDD Compilation Techniques for Feature Models. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 209–216.

[20] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. 2018. Efficient Sampling of SAT Solutions for Testing. In *Proc. Int'l Conf. on Software Engineering (ICSE).* ACM, 549–559.

[21] Kevin Feichtinger, Chico Sundermann, Thomas Thüm, and Rick Rabiser. 2022. It's Your Loss: Classifying Information Loss During Variability Model Roundtrip Transformations. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 67–78.

[22] David Fernández-Amorós, Sergio Bra, Ernesto Aranda-Escolástico, and Ruben Heradio. 2020. Using Extended Logical Primitives for Efficient BDD Building. *Mathematics* 8, 8 (2020), 1253:1–1253:17.

[23] Claudia Fritsch, Richard Abt, and Burkhardt Renz. 2020. The Benefits of a Feature Model in Banking. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, Article 9, 11 pages.

[24] Masahiro Fujita, Yusuke Matsunaga, and Taeko Kakuda. 1991. On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Logic Synthesis. In *Proc. Europ. Conf. on Design Automation (EURO-DAC).* IEEE, 50–54.

[25] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2019. Test Them All, Is It Worth It? Assessing Configuration Sampling on the JHipster Web Development Stack. *Empirical Software Engineering (EMSE)* 24, 2 (2019), 674–717.

[26] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. 2014. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines. *Trans. on Software Engineering and Methodology (TOSEM)* 40, 7 (2014), 650–670. doi:10.1109/TSE.2014.2327020

[27] Ruben Heradio, David Fernández-Amorós, José A. Galindo, David Benavides, and Don S. Batory. 2022. Uniform and Scalable Sampling of Highly Configurable Systems. *Empirical Software Engineering (EMSE)* 27, 2 (2022), 44.

[28] Tobias Heß, Tim Jannik Schmidt, Lukas Ostheimer, Sebastian Krieter, and Thomas Thüm. 2024. UnWise: High T-Wise Coverage From Uniform Sampling. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS).* ACM, 37–45.

[29] Tobias Heß, Sean Niklas Semmler, Chico Sundermann, Jacobo Torán, and Thomas Thüm. 2024. Towards Deterministic Compilation of Binary Decision Diagrams From Feature Models. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 136–147.

[30] Tobias Heß, Chico Sundermann, and Thomas Thüm. 2021. On the Scalability of Building Binary Decision Diagrams for Current Feature Models. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 131–135.

[31] Tobias Heß and Aaron Molt. 2025. *A Fast Counting-Free Algorithm for Computing Atomic Sets in Feature Models.* Technical Report arXiv:2501.12490. arXiv.

[32] Nagisa Ishiura, Hiroshi Sawada, and Shuzo Yajima. 1991. Minimization of Binary Decision Diagrams Based on Exchanges of Variables. In *Proc. Anual Conf. on Design Automation (DAC).* ACM, 472–473.

[33] Christian Kästner, Paolo G. Giarrusso, Tillmann Rendel, Sebastian Erdweg, Klaus Ostermann, and Thorsten Berger. 2011. Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In *Proc. Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA).* ACM, 805–824.

[34] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is There a Mismatch Between Real-World Feature Models and Product-Line Research?. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE).* ACM, 291–302.

[35] Sebastian Krieter. 2020. Large-Scale T-Wise Interaction Sampling Using YASA. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 29:1–29:4.

[36] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. 2020. YASA: Yet Another Sampling Algorithm. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS).* ACM, Article 4, 10 pages.

[37] Dominik Michael Krupke, Ahmad Moradi, Michael Perk, Phillip Keldenich, Gabriel Gehrke, Sebastian Krieter, Thomas Thüm, and Sándor Fekete. 2025. How Low Can We Go? Minimizing Interaction Samples for Configurable Systems. *Trans. on Software Engineering and Methodology (TOSEM)* (2025). To appear.

[38] Daniel Lohmann, Fabian Scheler, Reinhard Tartler, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. 2006. A Quantitative Analysis of Aspects in the eCos Kernel. *ACM SIGOPS Operating Systems Review* 40, 4 (2006), 191–204.

[39] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Evolution of the Linux Kernel Variability Model. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* Springer, 136–150.

[40] Chuan Luo, Binqi Sun, Bo Qiao, Junjie Chen, Hongyu Zhang, Jinkun Lin, Qingwei Lin, and Dongmei Zhang. [n. d.]. LS-Sampling: An Effective Local Search Based Sampling Approach for Achieving High T-Wise Coverage. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE).* ACM, New York, NY, USA, 1081–1092.

[41] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. 2016. A Comparison of 10 Sampling Algorithms for Configurable Systems. In *Proc. Int'l Conf. on Software Engineering (ICSE).* ACM, 643–654.

[42] Jeho Oh, Paul Gazzillo, and Don Batory. 2019. t-wise Coverage by Uniform Sampling. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 84–87.

[43] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Maggie Myers. 2019. *Uniform Sampling From Kconfig Feature Models.* Technical Report TR-19-02. University of Texas at Austin, Department of Computer Science.

[44] Justyna Petke, Myra B. Cohen, Mark Harman, and Shin Yoo. 2015. Practical Combinatorial Interaction Testing: Empirical Findings on Efficiency and Early Fault Detection. *IEEE Trans. on Software Engineering (TSE)* 41, 9 (2015), 901–924. doi:10.1109/TSE.2015.2421279

[45] Tobias Pett, Tobias Heß, Sebastian Krieter, Thomas Thüm, and Ina Schaefer. 2023. Continuous T-Wise Coverage. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 87–98.

[46] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product Sampling for Product Lines: The Scalability Challenge. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC).* ACM, 78–83.

[47] Quentin Plazar, Mathieu Acher, Gilles Perrouin, Xavier Devroey, and Maxime Cordy. 2019. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet?. In *Proc. Int'l Conf. on Software Testing, Verification and Validation (ICST).* IEEE, 240–251.

[48] Alexander Von Rhein, JöRG Liebig, Andreas Janker, Christian Kästner, and Sven Apel. 2018. Variability-Aware Static Analysis at Scale: An Empirical Study. *ACM*

*Trans. Softw. Eng. Methodol.* 27, 4, Article 18 (Nov. 2018), 33 pages. doi:10.1145/3280986

[49] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems. In *Proc. Int'l Conf. on Automated Software Engineering (ASE)*. IEEE, 342–352.

[50] Sergio Segura. 2008. Automated Analysis of Feature Models Using Atomic Sets. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*, Vol. 2. IEEE, 201–207.

[51] Fabio Somenzi. 2020. CUDD. Website: https://github.com/vscosta/cudd. Accessed: 2020-06-13.

[52] Chico Sundermann, Vincenzo Francesco Brancaccio, Elias Kuiter, Sebastian Krieter, Tobias Heß, and Thomas Thüm. 2024. Collecting Feature Models from the Literature: A Comprehensive Dataset for Benchmarking. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 54–65.

[53] Chico Sundermann, Tobias Heß, Michael Nieke, Paul Maximilian Bittner, Jeffrey M. Young, Thomas Thüm, and Ina Schaefer. 2023. Evaluating State-of-the-Art #SAT Solvers on Industrial Configuration Spaces. *Empirical Software Engineering (EMSE)* 28, 29 (2023), 38.

[54] Chico Sundermann, Elias Kuiter, Tobias Heß, Heiko Raab, Sebastian Krieter, and Thomas Thüm. 2023. On the Benefits of Knowledge Compilation for Feature-Model Analyses. *Annals of Mathematics and Artificial Intelligence (AMAI)* 92, 5 (2023), 1013–1050.

[55] Chico Sundermann, Michael Nieke, Paul Maximilian Bittner, Tobias Heß, Thomas Thüm, and Ina Schaefer. 2021. Applications of #SAT Solvers on Feature Models. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, Article 12, 10 pages.

[56] Chico Sundermann, Heiko Raab, Tobias Heß, Thomas Thüm, and Ina Schaefer. 2024. Reusing d-DNNFs for Efficient Feature-Model Counting. *Trans. on Software Engineering and Methodology (TOSEM)* 33, 8, Article 208 (2024), 32 pages.

[57] Reinhard Tartler, Daniel Lohmann, Christian Dietrich, Christoph Egger, and Julio Sincero. 2012. Configuration Coverage in the Analysis of Large-Scale System Software. *ACM SIGOPS Operating Systems Review* 45, 3 (2012), 10–14.

[58] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 6:1–6:45.