UnWise: High T-Wise Coverage from Uniform Sampling

Tobias Heß
University of Ulm
Germany

Tim Jannik Schmidt
University of Ulm
Germany

Lukas Ostheimer
University of Ulm
Germany

Sebastian Krieter
University of Ulm
Germany

Thomas Thüm
University of Ulm
Germany

ABSTRACT

Configuration spaces of industrial product lines are typically too large to be tested exhaustively. Therefore, testing in practice is often carried out on samples, sets of configurations which satisfy the requirements of the testing scenario. For t-wise sampling, the objective is to cover all t-wise interactions between configurable options with as few configurations as possible. However, a trade-off needs to be made between t, sampling time, sample size, and achieved coverage. In addition, it is infeasible for larger systems to even compute the set of all 2-wise interactions in practicable time. In this work, we reevaluate the performance of uniform samplers in terms of 2-wise coverage and come to a more positive result than previous research. We also present completion and reduction algorithms that greatly improve said performance. As a baseline for comparison, we additionally evaluate the two state-of-the-art dedicated t-wise samplers Baital and YASA. In doing so, we are the first to evaluate and compare these samplers on a large set of industrial feature models.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

Uniform Sampling, T-Wise Sampling, Feature-Model Analysis, Industrial Feature Models

ACM Reference Format:
Tobias Heß, Tim Jannik Schmidt, Lukas Ostheimer, Sebastian Krieter, and Thomas Thüm. 2024. UnWise: High T-Wise Coverage from Uniform Sampling. In *18th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2024), February 07–09, 2024, Bern, Switzerland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3634713.3634716

1 INTRODUCTION

Contemporary industrial product lines commonly have thousands of configurable options (i.e., *features*) which give way to enormous configuration spaces (in recent work we encountered models with up to 10^{1500} valid configurations [34]). As this makes exhaustive testing infeasible [13], sampling techniques are used to create sets of representative configurations of manageable size (i.e., *samples*) [32]. For purposes such as combinatorial interaction testing [7, 24, 25], one is interested in covering as many interactions as possible with the configurations in the sample. A t-wise coverage of 100 % means that every interaction between t features is present in at least one configuration in the sample (i.e., *covered*).

A number of specialized samplers have been developed with the goal of achieving high to full t-wise coverage with as few configurations as possible [2, 4, 17, 22, 29]. However, their time requirements [2, 21, 22, 29], yielded sample sizes [4, 29], or achieved coverages [4, 29] commonly make them impractical when applied to real-world feature models [31, 32]. For example, the state-of-the-art sampler YASA is capable of sampling hard models such as `Linux 2.6.28` [34, 36] within 30 minutes [22], achieves 100 % 2-wise coverage, but requires more than 500 configurations to reach this coverage. For large but underconstrained models, such as `Automotive02`, YASA fails to scale, as it, in essence, needs to iterate over all valid 2-wise interactions and therefore takes days [22].

For scenarios such as continuous integration, where time is at the essence and resources are limited, both a sampling time of 30 minutes and sample size of 500 configurations are impractical. In the case of JHipster, an open-source developer framework for web applications and microservices, there are only enough resources available to test 12 configurations per commit [13]. In 2016, more than 36,000 kernels were build per day for testing the Linux kernel [31], resulting in 25 configurations being tested per commit on average.[1] Therefore, we propose to trade-off coverage for increased sampling speed and reduced sample size, which in turn lowers the time and resource requirements for testing.

In this work, we revisit work by Oh et al. and evaluate the coverages achieved by uniform sampling [29]. However, in contrast to their work, our approach achieves high (98 % on median) 2-wise coverages on all models to which uniform sampling, in this case the sampler Spur [1], scales. We achieve this by applying two post-processing steps to the uniformly generated sample. First, we utilize a SAT solver to generate configurations that cover uncovered literals and thereby *complete* the sample until we reach 100 % 1-wise coverage. Second, we apply a coverage-maintaining *reduction* to the sample, reducing its size, while maintaining 100 % 1-wise coverage. Both post-processing steps are typically facilitated in under one second. Alternatively, we also developed a more time-intensive reduction that maintains the 2-wise coverage, which we use to select the best n configurations, in terms of coverage, from a sample.

[1] https://people.netfilter.org/hawk/presentations/ifdef2016/ifdef_FOSD2016.pdf

VaMoS 2024, February 07–09, 2024, Bern, Switzerland

ACM ISBN 979-8-4007-0877-0/24/02.
https://doi.org/10.1145/3634713.3634716

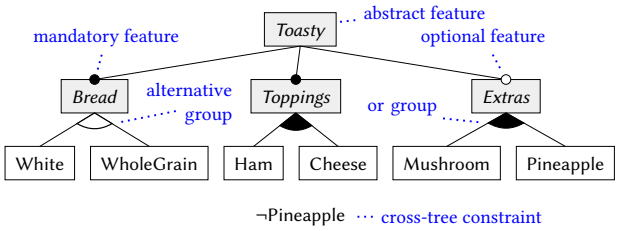

Figure 1: Running Example of a Feature Model

In our evaluation, we evaluate our approach on 49 real-world feature models (Oh et al. evaluated only FinancialServices [12, 29]) and compare it to the state-of-the-art t-wise samplers Baital [3, 4] and YASA [22]. In doing so, we are the first to evaluate the performance of Baital and YASA on a large set of industrial feature models. To investigate the influence of the sample, we compare the uniform sampler SPUR [1] against the non-uniform [14] sampler Quicksampler [10]. In particular, we investigate the following research questions:

RQ1 How well do state-of-the-art uniform and t-wise samplers scale to real-world feature models?
RQ2 What t-wise coverages are achieved by sampling uniformly?
RQ3 What t-wise coverages are achieved by samples of practical size?

The remainder of this work is structured as follows. We give background information in Section 2 and outline our sampling method in Section 3. We present the findings of our evaluation in Section 4 and compare it with previous works in Section 5.

2 BACKGROUND

In this section, we give a brief introduction to feature models, their analysis as well as uniform and t-wise sampling.

2.1 Feature Models

Feature models have emerged as the default for modeling the variability in configurable systems [5, 6]. Configurable options are modeled as features, which are hierarchically organized in a feature diagram. Additionally, cross-tree constraints can impose additional constraints on the set of valid configurations.

Consider Figure 1, which contains our running example, a product-line for tasty toasties. Abstract features are used to organize features but do not impact derived products [37]. Selecting a child features mandates selecting the parent feature as well. Likewise, mandatory features must be selected when the parent is selected. In or groups, at least one feature must be selected when the parent feature is selected and precisely one feature in alternative groups. Finally, the root feature must always be selected.

Let $C(M)$ denote the set of all valid configurations of a feature model M. Every configuration in $C(M)$ needs to satisfy the constraints imposed by the feature diagram and the cross-tree constraints. For example, $C_1 = \{\text{White}, \text{Ham}, \text{Cheese}\}$ is a valid configuration for our running example, while $C_2 = \{\text{White}, \text{Pineapple}\}$ is invalid. In particular, C_2 does neither contain Ham nor Cheese, violating the or group of Toppings, even though Toppings is selected (as it is a mandatory feature of the root feature). Furthermore, Pineapple must not be selected, as its selection violates the cross-tree constraint.

Features like Pineapple that are not part of any valid configuration are called dead features. Likewise, features that appear in every valid configuration are called core features [6]. While models like our running example can be analyzed by hand, this is infeasible for larger models [6]. Therefore, feature models are translated into Boolean formulas [28] which are then analyzed using, for instance, SAT solvers [6, 28].

2.2 t-Wise Sampling

The goal of t-wise sampling is to cover all t-wise interactions between features. For $t = 2$ the theoretically possible interactions between features x and y are (x, y), $(\neg x, y)$, $(x, \neg y)$, and $(\neg x, \neg y)$. However, not all of this interactions must be valid, due to constraints in the feature model.

Let M be a feature model with a set of non-core, and non-dead features $\mathbb{F}^*(M)$, inducing a set of literals

$$\mathbb{L} = \{l \mid l = f \text{ or } l = \neg f \text{ for } f \in \mathbb{F}^*\}$$

Every literal $l \in \mathbb{L}$ thus represents the selection or deseletion of a feature. In addition, let $F = F(M)$ be a Boolean formula encoding M [6]. With this, we may define the set of all possible ordered t-wise interactions as

$$\mathbb{I}_t(F) = \{(l_{i_1}, \ldots, l_{i_t}) \in \mathbb{L}^t \mid i_1 < \ldots < i_t\}$$

and the subset of all valid interactions as

$$\mathbb{I}_t^*(F) = \left\{I \in \mathbb{I}_t \;\middle|\; \text{SAT}(F \wedge I)\right\}$$

where $F \wedge I = F \wedge \bigwedge_{l \in I} l$. Lastly, let $S \subset C$, then $\mathbb{I}_t(F, S) \subset \mathbb{I}_t(F)$ denotes the set of interactions that appear in at least one configuration $c \in S$, i.e.:

$$\mathbb{I}_t(F, S) = \{I \in \mathbb{I}_t(F) \mid \exists\, c \in S \text{ with } I \in c\}$$

The t-wise coverage achieved by a sample $S \subset C$ is defined as the ratio

$$\text{cov}_t(F, S) = \frac{|\mathbb{I}_t(F, S)|}{|\mathbb{I}_t^*(F)|} \quad [22]$$

Example 2.1. Let M be a feature model with $\mathbb{F}^*(M) = \{x, y, z\}$ and $F(M) = (x \vee y) \wedge (\neg x \vee \neg y)$. Then

$$\mathbb{I}_2^*(F) = \{(x, \neg y), (x, z), (x, \neg z), (\neg x, y), (\neg x, z), (\neg x, \neg z), (y, z), (y, \neg z)\}$$

For $S = \left\{\{x, \neg y, z\}, \{\neg x, y, \neg z\}\right\}$, the set of interactions covered by S is

$$\mathbb{I}_2(F, S) = \{(x, \neg y), (x, z), (\neg y, z), (\neg x, y), (\neg x, \neg z), (\neg y, \neg z)\}$$

Therefore $\text{cov}_2(F, S) = \frac{6}{8} = 75\,\%$.

2.3 Uniform Sampling

In uniform sampling, configurations are repeatedly chosen uniformly at random (i.e, drawing with replacement without order) until a specified sample size m is reached [14]. If we assume that there are n valid configurations in total, then the probability for a configuration c to be included in a sample $S \subset C$ of size m is

$$\mathbb{P}(c \in S) = 1 - \left(\frac{n-1}{n}\right)^m \tag{1}$$

Given a reasonable sample size m, uniform sampling maintains certain characteristics of the feature model, such as the commonality (i.e., the ratio of configurations in the sample containing a feature converges to the commonality of the feature for large enough sample sizes). Consequently, if a feature does not appear in a uniform sample of non-trivial size, it will also be rarely selected in practice, as it only appears in few configurations. In short, configurations in uniform samples are statistically representative of the configuration space.

Based on Equation 1, one can predict the t-wise coverage achieved by a uniform sample [29]. Let $I \in \mathbb{I}_t^*(F)$ be a t-wise interaction and let $n(I)$ be the number of valid configurations including the interaction I (i.e., $n(I) = |\{c \in C \mid I \in c\}|$). Then Equation 2 denotes the probability that a uniform sample of size m covers the interaction I.

$$\mathbb{P}(\exists\, c \in S : I \in c) = 1 - \left(\frac{n-n(I)}{n}\right)^m \tag{2}$$

3 OUR APPROACH

In this section, we present our approach for achieving high t-wise coverages by post-processing uniform samples. First, we calculate a sample using a uniform sampler. Second, we measure the 1-wise coverage of the sample, using a SAT solver (in our case MiniSAT [11], the PySAT [16] default). Third, we complete the sample by computing additional configurations with the SAT solver until all literals are covered (i.e., 100 % 1-wise coverage is achieved). Fourth and last, we reduce the sample while maintaining the 100 % 1-wise coverage, as described in Section 3.2. Alternatively, we can reduce the sample while maintaining the achieved 2-wise coverage of the sample.

The post-processing is, however, not limited to uniform samples and can also be applied to the samples generated by the t-wise samplers. In particular, the 2-wise reduction algorithm can be used to select a sub sample of fixed size from a sample.

3.1 Completion

As the commonality of a feature directly translates to the appearance of its literals in the uniform sample (cf. Section 2.3) it is likely that only one of the feature's literals appears in configurations. As an uncovered literal directly causes all t-wise interactions including this literal to be uncovered, even a 1-wise coverage of 99 % has the potential to snowball to abysmal t-wise coverages for $t \geq 2$.

Therefore, our approach harnesses the high performance of SAT solvers for feature models [23, 28] to generate additional configurations, until the sample achieves 100 % 1-wise coverage. Algorithm 1 depicts the completion procedure. For all literals that are not covered by the current sample, we call the SAT solver. As we already know that a configuration exists for an uncovered literal l (as else it would not be in the set of valid literals $\mathbb{I}_1^*$), we are not interested in the solver's decision, but rather the configuration it produces as proof of its decision. As other uncovered literals may appear in this configuration, we not only remove l but all newly covered literals from the set of uncovered literals.

Algorithm 1: 1-wise Completion

Input: formula F, sample S, $\mathbb{I}_1(F,S)$, $\mathbb{I}_1^*(F)$
Output: sample S' with $\mathbb{I}_1(F,S') = \mathbb{I}_1^*(F)$

```
S' ← S
uncovered ← 𝕀₁*(F) \ 𝕀₁(F, S)
foreach l ∈ uncovered do
    config ← solve SAT(F ∧ l)
    S' ← S' ∪ {config}
    uncovered ← uncovered \ config
end
return S'
```

After completion, our sample does achieve 100 % 1-wise coverage but likely contains redundant configurations that cover literals which are also covered by other configurations. While these configurations might cover additional t-wise interactions (cf. Section 4), we prioritize a smaller sample size over coverage for now.

3.2 Reduction

Our procedure to reduce the sample size is straightforward and aims at being fast and "good enough" over producing samples of minimal size. As uniform sampling already produces non-minimal samples in terms of coverage, even a sample of minimal size is likely to be larger than samples produced by dedicated t-wise samplers that aim for 100 % 1-wise coverage.

The procedure itself, which is depicted in Algorithm 2 incrementally selects configurations from the sample that cover the most literals or interactions currently not covered by configurations in the reduced sample. While this could be augmented to account for the rarity of literals or interactions appearing in configurations, we found in preliminary experiments that the improvement was marginal at best.

Algorithm 2: t-wise Reduction

Input: sample S
Output: S' with $\mathbb{I}_t(F,S) = \mathbb{I}_t(F,S')$ & $|S'| \leq |S|$

```
to_cover ← 𝕀_t(F, S)
S' ← ∅
while to_cover ≠ ∅ do
    foreach config ∈ S \ S' do
        value ← |{I ∈ to_cover | I ∈ to_cover}|
        if value ≥ value_best then
            config_best ← config
            value_best ← value
        endif
    end
    to_cover ← to_cover \ config_best
    S' ← S' ∪ {config_best}
end
return S'
```

Naturally, while the implementation of 1-wise reduction can follow Algorithm 2, already Line 1 is often infeasible for larger models and $t \geq 2$. Therefore, we exploit the following observation in our implementation for $t = 2$. Let $S' = \{A\}$ and we want to compute the number

$$\Delta = |\mathbb{I}_2(F, S' \cup \{B\})| - |\mathbb{I}_2(F, S')|$$

of newly covered 2-wise interactions by adding a valid configuration B to S'. Then it holds that

$$\Delta(B, A) = |B \cap A| \cdot |B \setminus A| + \frac{|B \setminus A| \cdot (|B \setminus A| - 1)}{2} \tag{3}$$

Clearly, interactions build from literals in $B \cap A$ are already covered by A and interactions build from $B \setminus A$ are not. Adding B to S' would cover all interactions in $(B \cap A) \times (B \setminus A)$, hence the first part of Equation 3 and literals in $B \setminus A$ also form interactions with each other, hence the second part (division by two to ignore permutations). All these interactions are trivially valid, as A and B were valid configurations.

Now let $S' = \{A, B\}, U = A \cup B$ and we want to compute $\Delta(C, U)$, i.e., the number of additionally covered interactions by adding a configuration C to S'. Now, Equation 3 does not necessarily hold anymore, as for instance both literals of a variable may have appeared in A or B and therefore, new interactions are also possible between literals in the set $C \cap U$. However, any interaction (x, y) (or (y, x) if $|x| \leq |y|$) newly covered by C must either stem from $x \in (C \cap A) \setminus B$ and $y \in (C \setminus A \setminus B) = (C \setminus U)$ or $x, y \in (C \setminus U)$. Our 2-wise reduction algorithm exploits these observations to replace Lines 1 and 5 in Algorithm 2.

4 EVALUATION

In this section, we present the findings of our evaluation and answer our research questions.

4.1 Preliminaries

We start by introducing the samplers we used in our evaluation, followed by the subject systems, the execution environment and our methodology.

4.1.1 Samplers. We use the two state-of-the-art t-wise samplers Baital and YASA, which to the best of our knowledge never have been jointly evaluated, to establish baseline coverages. In addition, we evaluate the three uniform samplers Smarch, Spur, and Quicksampler.

Baital. Baital [3, 4] is a sampling framework that generates samples with high t-wise coverage based on different literal-weighting strategies. As suggested in its documentation,[2] we use approximate heuristics for its preprocessing and coverage estimation.

YASA. YASA [21, 22] is a greedy SAT-solver based t-wise sampling algorithm that allows to trade of sampling time vs sample size. We augmented its implementation in FeatureIDE [27] to allow for a graceful timeout which saves the current sample.

[2] https://github.com/meelgroup/baital

Table 1: Sampler Performance

		Runtime for Successes (s)			
Sampler	# Successes	min	max	med	avg
Uniform Samplers, t/o = 5 minutes					
Quicksampler	49	0.04	211.99	0.81	11.59
Smarch	4	25.32	156.52	145.02	117.97
Spur	45	0.01	187.67	0.75	14.82
t-wise Samplers, t/o = 5 minutes					
Baital (t = 1)	49	0.35	26.85	1.30	3.11
Baital (t = 2)	49	0.38	28.99	1.94	4.13
Yasa (t = 1)	48	0.69	t/o	4.38	17.39
Yasa (t = 2)	48	0.51	t/o	15.25	53.03

Smarch. Smarch [30] builds a uniform sample by recursively invoking the #SAT solver sharpSAT [38] based on partitioning the configuration space with the cube-and-conquer method by Heule et al. [15]. While Smarch is known to scale poorly to most industrial feature models [14], we include it for comparison to the work by Oh et al. [29].

Spur. Spur [1], uses Knuth's partitioning algorithm to partition the search space and, like Smarch, invokes sharpSAT [38] to compute the uniform samples.

Quicksampler. Quicksampler [10] computes samples probabilistically based on atomic mutations. However, both the sample's uniformity and the validity of the configurations are not guaranteed but only statistically probable [10, 14].

4.1.2 Subject Systems. For our evaluation, we use 49 real-world feature models from a variety of origins and domains. 36 models are provided by Oh et al. [30], containing models from the embedded and software systems domain. Furthermore, we include three representative models (minimum, median, and maximum number of features) from the large number of very similar CDL models [35], seven KConfig models, two automotive models [18] and one model from the financial domain [12]. 23 of the 48 models have less than 1,000 features, 21 models have between 1,000 and 10,000 features, and the remaining four models (`automotive02v4` (18,616 features), `embtoolkit-smarch` (23,516 features), `freetz` (31,012 features), `uclinux-config` (11,254 features) have up to 31,012 features.

4.1.3 Execution Environment. The evaluation was executed on a dedicated server with two Intel(R) Xeon(R) CPU E5-2620v3 2.4 GHz and 256 GB RAM under Ubuntu 22.04.

4.1.4 Methodology.

Uniform Sampling. For every subject systems, we attempted to compute a uniform sample of size $1024 = 2^{10}$ with each of the three uniform samplers Smarch, Spur, and Quicksampler with a timeout of five minutes. We chose this sample size based on previous work, and the sample sizes yielded by Baital and YASA in preliminary experiments. On success we removed duplicate configurations from the respective samples, which sporadically appear for small models. For Quicksampler, which does not guarantee the validity of the configurations in the sample, we additionally removed invalid configurations.

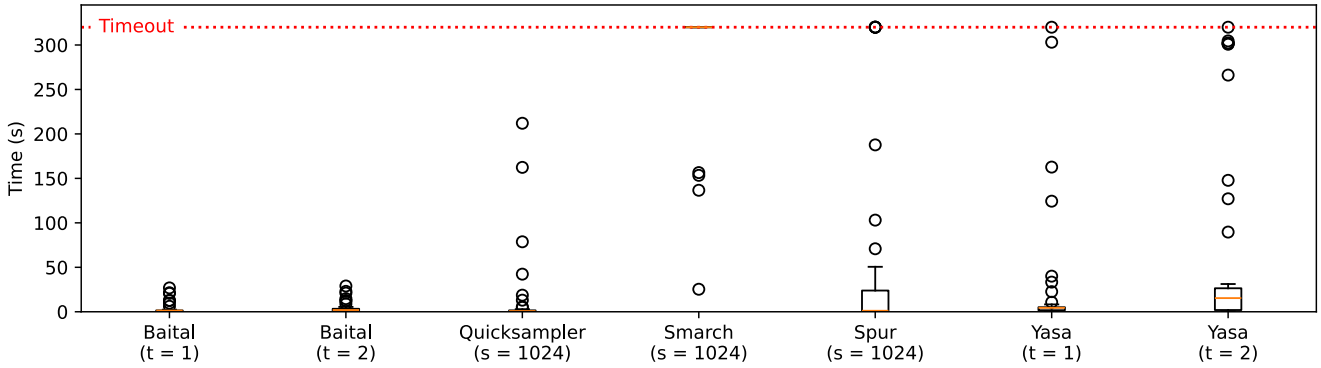

Figure 2: Sampling Times for Uniform and t-Wise Samplers (Timeout: 5 min)

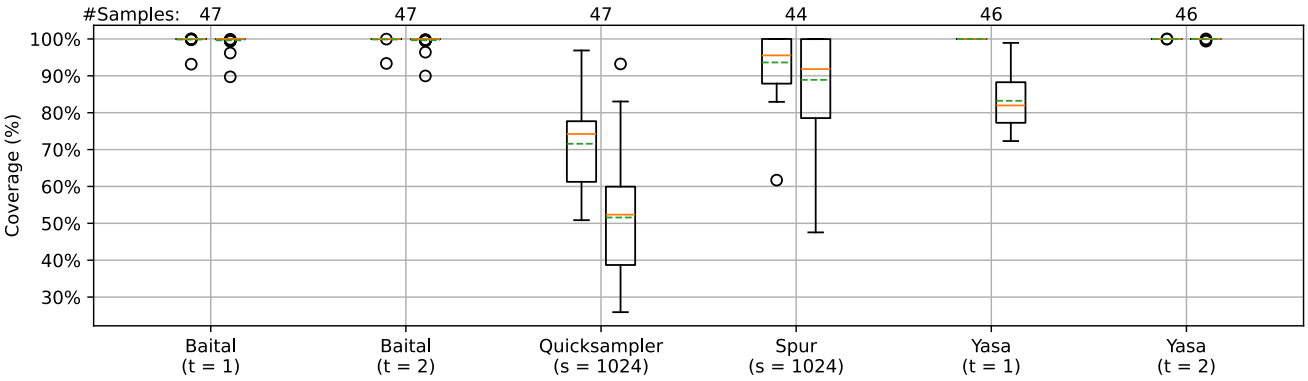

Figure 3: Achieved 1-wise (left) and 2-wise (right) coverages without post-processing.

t-Wise Sampling. As a baseline, we used Baital and YASA to compute 1- and 2-wise samples for all of the subject systems within time limits of five minutes and a time limit of one hour.

Coverage Calculation. Based on the samples yielded from both uniform and t-wise sampling, we measured the achieved 1- and 2-wise coverages. To investigate the impact of our post-processing, we additionally measured the achieved coverages and sample sizes after 1-wise completition together with no reduction, 1-wise reduction, and 2-wise reduction.

4.2 Results

In the following, we present the results of our experiments, grouped by the overall sampler performance in terms of number of successes, sampling time, and sample size, as well as the achieved coverages with and without post-processing.

4.2.1 Sampling Performance. Out of the three uniform samplers in our evaluation, only Quicksampler (QS) was capable to compute samples for all of the 49 models within the time limit of 5 minutes. Spur performed second best but was unsuccessful for the models `buildroot` (QS: 78.7 s), `embtoolkit-smarch` (QS: 162.4 s), `freetz` (QS: 212.0 s), and `linux-2.6.33.3` (QS: 13.0 s). Spur was additionally able to compute a sample for `embtoolkit-smarch` (612.7 s), within a timeout of 1 hour.

Smarch, which was used as the uniform sampler by Oh et al. [29] in their measurement of t-wise coverage from uniform sampling, was only able to sample four trivial models. For example, Smarch required 230.6 s to compute 1024 sample configurations for `JHipster` [13], which was sampled by each of the other uniform samplers in well beyond 1 s, respectively. Therefore, we exclude Smarch from further consideration.

Both t-wise samplers, Baital and YASA are capable of producing intermediate results. Therefore, we count the availability of any non-empty sample at timeout as success. With the configuration from above, Baital computed 1- and 2-wise samples for all of the 49 models within 30 s, respectively. Within the time limit of 5 minutes, YASA was able to sample all models but `embtoolkit-smarch` (both 1- and 2-wise), but was successful within 1 hour.

Without post-processing, YASA outperforms Baital significantly in terms of median sample size (69.5 vs 500[3]), average sample size

[3]Default sample size limit of Baital

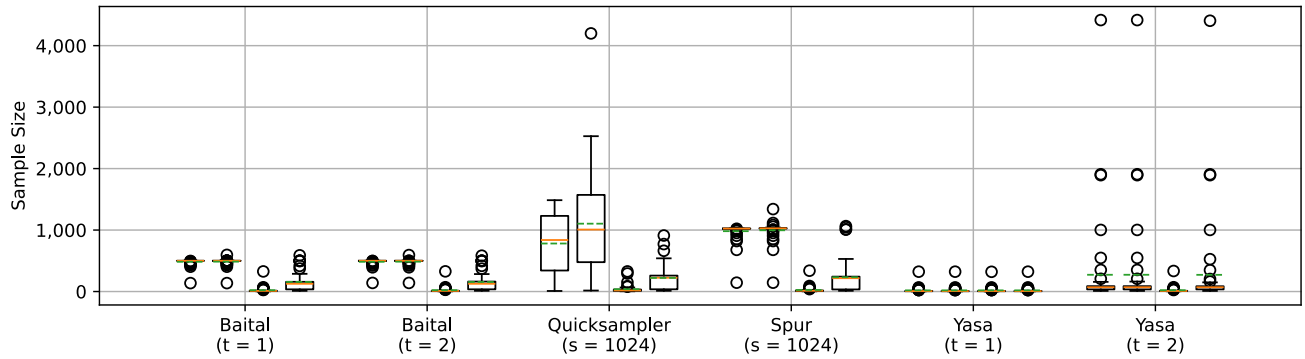

Figure 4: Sample Sizes (no post-processing, 1-wise compl., 1-wise compl. + 1-wise red., 1-wise compl. + 2-wise red.)

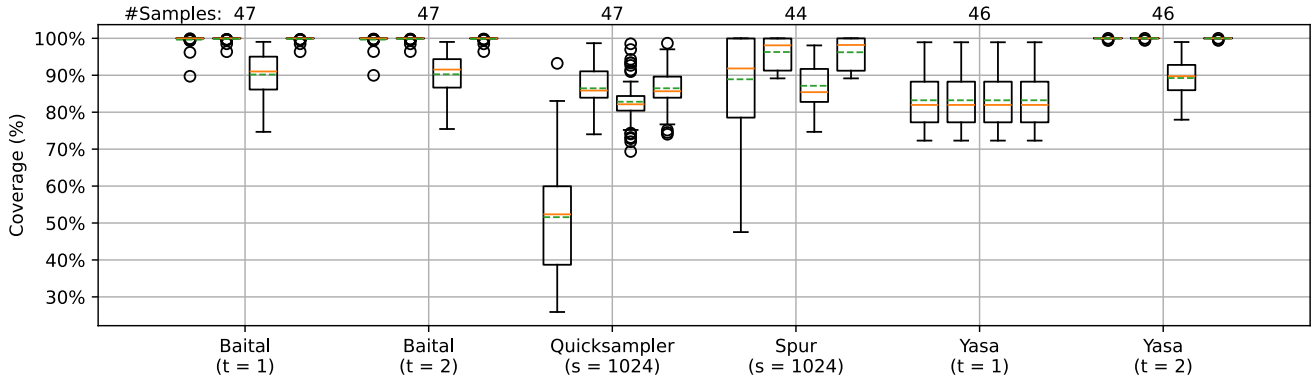

Figure 5: 2-wise Coverages (no post-processing, 1-wise compl., 1-wise compl. + 1-wise red., 1-wise compl. + 2-wise red.)

(272.5[4] vs 483.8), and number of models for which 100 % 2-wise coverage was achieved (43 vs 16).

4.2.2 Achieved Coverages. We were unable to count the number of valid 2-wise interactions for `automotive02v4` and `freetz` within 24 hours and can therefore make no statement on the achieved coverages on these models. Figure 3 depicts the 1- and 2-wise coverages achieved by samples by the respective samplers without any post-processing.

The best results are achieved by 2-wise sampling with YASA, which achieved 100 % 2-wise coverage for all but three models, namely `linux-2.6.33.3` (99.40 %), `automotive01` (99.99 %), and `buildroot` (99.56 %). For Baital, the results are curious, as both 1-wise and 2-wise sampling appears to perform virtually equal with regards to 2-wise coverage. This is most likely due to its prioritization of sampling time over sample size and coverage.

For the two other samplers, Quicksampler and Spur, the difference between sampling non-uniformly and sampling uniformly are very apparent in the achieved coverages, even without taking post-processing into account. With a target sample size of 1024, Quicksampler achieves a 2-wise coverage of 52.3 % on median and 51.6±16.3 % on average and does not achieve 100 % 2-wise coverage for any of the models. Spur on the other hand, achieves a median of 91.8 % and a mean of 88.9 ± 11.7 %, and nine times with 100 % 2-wise coverage.

4.2.3 Post-Processing. So far, we looked at the time and achieved coverages, without consideration of the sample sizes. Figure 4 depicts the sample sizes without post-processing, after 1-wise completion, and after 1-wise completion and 1-wise or 2-wise reduction, respectively. Starting again with the t-wise samplers, we see that without post-processing, YASA (t = 2) on median and average outperforms Baital significantly in terms of sample size. However, for four models, YASA requires more than 500 configurations to reach its target coverage of 100 %. The difference in sample size becomes even more apparent when comparing the sample size of YASA (t = 1) to Baital (t = 1), where Baital requires our post-processing to even come close to the sample sizes generated by YASA for full 1-wise coverage. The sample sizes of both are only marginally affected by the completion, due to their already high to full 1-wise coverages.

For the uniform samplers, Quicksampler produces more than the targeted 1,024 configurations for some models and also commonly loses 20 % or more due to duplicate or invalid configurations. Spur only loses configurations due to duplicates and always has a sample

[4] 67.5 if one ignores the 5 models with sample sizes above 500, as YASA prioritizes 100 % coverage over sample size

Table 2: Sample Size and Average 2-Wise Coverage ± Standard Deviation for Different Post-processing Scenarios

Sampler	No post-processing		Compl. + 2-wise Red.		Compl. + 2-wise Red. to size 16
	Size	Cov	Size	Cov	Cov
Baital (t = 1)	484±55.8	99.63±1.6 %	156±155.3	99.83±0.6 %	93.45±8.0 %
Baital (t = 2)	484±55.9	99.65±1.5 %	156±154.7	99.83±0.6 %	93.42±8.0 %
Quicksampler	781±500.4	51.58±16.3 %	221±211.9	86.48±5.9 %	81.70±8.6 %
Spur	982±145.2	88.91±11.7 %	233±285.8	96.23±4.0 %	88.76±9.1 %
Yasa (t = 1)	19±48.2	83.23±6.9 %	19±48.2	83.23±6.9 %	80.88±6.7 %
Yasa (t = 2)	272±744.7	99.98±0.1 %	272±743.2	99.98±0.1 %	93.21±8.2 %

size of or below 1,024. While Figure 3 already suggested a negative impact from the missing uniformity of its sample, this is illustrated again by the fact that the increase in sample size due to 1-wise completion is more sizable for Quicksampler than for Spur.

Figure 5 depicts the 2-wise coverages achieved by the samplers after the various post-processing scenarios have applied. The leftmost column per sampler configuration (i.e., no post-processing) is identical to the right column in Figure 3. One can see, that all samplers except YASA benefit from 1-wise completion. Quicksampler benefits the most (med: 52.4 % → 85.9 %, avg: 51.6±16.3 % → 86.5±6.0 %) and Spur second most (med: 91.8 % → 98.1 %, avg: 88.9±11.7 % → 96.3±3.9 %).

Furthermore, while we see that 1-wise reduction is too aggressive and reduces the 2-wise coverage significantly, the resulting coverages after 1-wise reduction (cf., 3rd column in Figure 5) are with 6 % for all samplers, except Quicksampler. On the other hand, all samplers but YASA, benefit from 2-wise reduction, but are still outperformed by YASA, which has the smallest sample sizes, with the best 2-wise coverage, for all models to which it scales.

4.2.4 Practical Sample Sizes. Table 2 contains the sizes and achieved 2-wise coverages achieved by no post-processing, 1-wise completion and 2-wise reduction, and 1-wise completion and 2-wise reduction to size 16. While or choice of target size 16 is mostly arbitrary, it aligns well with sample sizes currently used in practice [13]. We see that the 2-wise reduction significantly reduces the sample sizes of Baital and the uniform samplers, while leaving the sample size of YASA pretty much unaffected. Finally, by limiting the sample size to 16, we see that the dedicated 2-wise samplers still achieve coverages above 93 %.

4.3 Discussion

Sampler Scalability. With regard to RQ1 "*How well do state-of-the-art uniform and t-wise samplers scale to real-world feature models?*", our evaluation shows that all evaluated samplers except Smarch are capable of scaling to the vast majority of models. Quicksampler achieves its perfect scalability by sacrificing both the validity and uniformity [8, 10, 14, 33] of its samples. Curiously, while both Smarch and Spur depend on sharpSAT [38], Spur scales to all 45 models for which sharpSAT scales [34], while Smarch only scales to four models.

Sample Quality. With regard to RQ2 "*What t-wise coverages are achieved by sampling uniformly?*", we find that uniform sampling with state-of-the-art uniform sampler like Spur is capable of achieving high 2-wise coverages for all models but the non-software model `FinancialServices`. With our post-processing, the sample size can be reduced up to a factor of 4 and 2-wise coverages of 96.3 % on average are achieved. However, the large starting sample size of uniform sampling results in a reduction effort that typically exceeds the sampling time (we encountered 2-wise reduction times of up to 30 minutes for larger models). In short, it is always better to just use YASA with our timeout augmentation.

Practical Sampling. Our experiment with a fixed sample size of 16 (cf., Table 2), shows that high 2-wise coverages of over 93.5 % on average can still be achieved. Therefore we conclude as answer for RQ3 "*What t-wise coverages are achieved by samples of practical size?*" that limiting the sample size to meet resource limitations is possible without loosing too much coverage.

FinancialServices. As mentioned before, the non-software model FinancialServices constitutes a hard benchmark for t-wise sampling [4, 21, 22, 32], due to its excessive use of alternative groups and cross-tree constraints [12]. Using Equation 2, Oh et al. estimated that a uniform sample size of 10^{12} is required to achieve a 2-wise coverage of above 90% and a sample size of 10^{14} for 99.99 % [29]. As FinancialServices only contains about $9.7 \cdot 10^{13}$ configurations in total, this would de facto be enumeration. However, we found in previous research that 100 % 2-wise coverage of FinancialServices can be achieved with around 4,400 configurations [21]. Our evaluation confirms this (cf. Figure 4). In addition, we found that 2-wise coverage of 97.0 % can be achieved with 330 configurations by combining SPUR with our inexpensive 1-wise completion and 1-wise reduction.

4.4 Threats to Validity

Internal Validity. Due to probabilistic and therefore non-deterministic nature of the samplers, distinct runs may compute samples of different size, with different coverage, with different runtimes. However, in preliminary experiments on a subset of models, we found that the differences in outcome are very marginal and do not change the overall outcome. In their evaluation of Baital, Baranov et al. came to a similar observation after repeating their experiments [4]. Therefore, we decided against multiple repetitions of our experiment, due to large investment of time and resources. Nonetheless, our artifact[5] supports multiple repetitions.

In addition, our implementation of the wrapper for the samplers, the post-processing, or the coverage computation may be flawed. However, we verified for some smaller models that coverage achieved by uniform sampling with SPUR corresponds to the theoretical expected value(cf. Equation 2 [29]). Furthermore,

[5] https://zenodo.org/records/10303558

we successfully verified the calculated coverages for some systems with the values computed internally by Baital and YASA.

External Validity. It is possible that our approach yields different results for models not included in our evaluation. However, we chose a large number of models with different sizes and complexities from a variety of domains, including non-software systems. All of these models have been used in previous works [4, 14, 22, 34]. Our evaluation itself does contain a significant outlier in FinancialServices, which we therefore discussed separately in Section 4.3.

5 RELATED WORK

In this section, we summarize works on t-wise and uniform sampling and their relation to ours.

Closest to our work is the work of Oh et al. that measured the 1- and 2-wise coverages achieved by samples generated with uniform sampling [29]. They used their uniform sampler Smarch [30] to compute samples of different sizes up to 10^{14} configurations for the `FinancialServices` [12] model and concluded that uniform sampling alone is not enough to achieve high to full 2-wise coverage. Our evaluation however shows that this is an outlier limited to `FinancialServices` which can be overcome by our post-processing (cf. Section 4.3) and that uniform sampling achieves a 2-wise coverage of 88.9 % without and 96.2 % with post-processing for all other models in our evaluation.

Like their and our work, both YASA [21, 22] and Baital [3, 4] are answers to the sampling scalability challenge of Pett et al. [32]. The challenge asks for submissions evaluating the t-wise sampling of three models (`FinancialServices`, `linux-2.6.33.3`, and `automotive02v4`) with regards to time and memory requirements. We contribute to this cause by being (to the best of our knowledge) the first to evaluate different strategies for t-wise sampling on a large number of industrial feature models. For instance, YASA and Baital have never been evaluated against each other, previously.

Heradio et al. evaluate a number of uniform samplers (including Quicksampler, Smarch, and Spur) in terms of sampling time and uniformity on nine industrial feature models. Our results regarding sampling time and sampling success are in line with theirs. While we are not primarily interested in the sample's uniformity, we attribute the coverage differences between Quicksampler and Spur to the lack of uniformity in Quicksampler's samples, as diagnosed by them [14].

Varshosaz et al. give an overview and classification of the literature regarding product sampling in the context of software product lines [39]. However all samplers of relevance to our work, like ICPL [20], IncLing [2], or based on Chvatal [9], are outperformed by an order of magnitude or more by YASA [22].

Finally, Medeiros et al. compared ten sampling approaches in terms of fault detection, recommending that t-wise sampling with high values of t is used for rigorous testing [26]. Similar observations were made by Halin et al. in their case study of JHipster [13]. While generating 3-wise samples with high to full coverage will be feasible for a large number of models in our evaluation, measuring their coverage presents a challenge as of now, due to the required effort of counting the number valid 3-wise interactions.

6 CONCLUSION

In this work, we compared the 1- and 2-wise coverages achieved by dedicated t-wise and uniform samplers on a large set of industrial feature models. We find that it is always beneficial to use our timeout-augmented version of YASA, as it produces the smallest samples with the highest 2-wise coverages. Nevertheless, we found that uniform sampling is capable of achieving much higher coverages than previously reported [29], even without our post-processing. With the post-processing presented in this work, both sample sizes and 2-wise coverages of practical value can be achieved by means of uniform sampling. For the future, we plan to explore $t > 2$, more sophisticated means of sample size reduction, and the behavior of samples on model evolution.

Artifact

The tooling of this work is available under https://zenodo.org/records/10303558.

REFERENCES

[1] 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, Berlin, Heidelberg, Germany, 135–147.

[2] 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)* (Amsterdam, Netherlands). ACM, New York, NY, USA, 144–155. https://doi.org/10.1145/2993236.2993253

[3] Eduard Baranov and Axel Legay. 2022. Baital: An Adaptive Weighted Sampling Platform for Configurable Systems. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 46–49. https://doi.org/10.1145/3503229.3547030

[4] 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)* (Virtual Event, USA). ACM, New York, NY, USA, 1114–1126. https://doi.org/10.1145/3368089.3409744

[5] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Springer, Berlin, Heidelberg, Germany, 7–20. https://doi.org/10.1007/11554844_3

[6] 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.

[7] Ivan Do Carmo Machado, John D. McGregor, Yguaratã Cerqueira Cavalcanti, and Eduardo Santana De Almeida. 2014. On Strategies for Testing Software Product Lines: A Systematic Literature Review. *J. Information and Software Technology (IST)* 56, 10 (2014), 1183–1199. https://doi.org/10.1016/j.infsof.2014.04.002

[8] Sourav Chakraborty and Kuldeep S. Meel. 2019. On Testing of Uniform Samplers. In *Proc. Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 7777–7784. https://doi.org/10.1609/aaai.v33i01.33017777

[9] Vasek Chvatal. 1979. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research (MOR)* 4, 3 (1979), 233–235.

[10] 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)*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, Boston, MA, USA, 549–559. https://doi.org/10.1145/3180155.3180248

[11] Niklas Eén and Niklas" Sörensson. 2004. An Extensible SAT-solver. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, Berlin, Heidelberg, Germany, 502–518.

[12] 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)* (Montreal, QC, Canada). ACM, New York, NY, USA, Article 9, 11 pages. https://doi.org/10.1145/3382025.3414946

[13] 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 (July 2019), 674–717.

[14] 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. https://doi.org/10.1007/s10664-021-10102-5

[15] Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. 2011. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. In *Proc. Int'l Haifa Verification Conf. (HVC) (Lecture Notes in Computer Science, Vol. 7261)*, Kerstin Eder, João Lourenço, and Onn Shehory (Eds.). Springer, 50–65. https://doi.org/10.1007/978-3-642-34188-5_8

[16] Alexey Ignatiev, António Morgado, and João Marques-Silva. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT) (Lecture Notes in Computer Science (LNCS), Vol. 10929)*. Springer, 428–437. https://doi.org/10.1007/978-3-319-94144-8_26

[17] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible. In *Proc. Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS)*. Springer, Berlin, Heidelberg, Germany, 638–652. https://doi.org/10.1007/978-3-642-24485-8_47

[18] 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)* (Paderborn, Germany). ACM, New York, NY, USA, 291–302. https://doi.org/10.1145/3106237.3106252

[19] Donald E. Knuth. 2009. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams.* Addison-Wesley, Boston, MA, USA.

[20] Matthias Kowal, Sandro Schulze, and Ina Schaefer. 2013. Towards Efficient SPL Testing by Variant Reduction. In *Proc. Int'l Workshop on Variability and Composition (VariComp)* (Fukuoka, Japan). ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/2451617.2451619

[21] 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.

[22] 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)* (Magdeburg, Germany). ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3377024.3377042

[23] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. 2015. SAT-Based Analysis of Large Real-World Feature Models Is Easy. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Springer, Berlin, Heidelberg, Germany, 91–100.

[24] Roberto E. Lopez-Herrejon, Stefan Fischer, Rudolf Ramler, and Aalexander Egyed. 2015. A First Systematic Mapping Study on Combinatorial Interaction Testing for Software Product Lines. In *Proc. Int'l Workshop on Combinatorial Testing (IWCT)*. IEEE, Washington, DC, USA, 1–10. https://doi.org/10.1109/ICSTW.2015.7107435

[25] John McGregor. 2010. Testing a Software Product Line. In *Testing Techniques in Software Engineering*. Springer, Berlin, Heidelberg, Germany, 104–140.

[26] 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)* (Austin, Texas). ACM, New York, NY, USA, 643–654. https://doi.org/10.1145/2884781.2884793

[27] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE.* Springer, Berlin, Heidelberg, Germany. https://doi.org/10.1007/978-3-319-61443-4

[28] Marcílio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. SAT-Based Analysis of Feature Models is Easy. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (San Francisco, California). Software Engineering Institute, Pittsburgh, PA, USA, 231–240.

[29] 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.

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

[31] 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. https://doi.org/10.1145/3579027.3608980

[32] 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)* (Paris, France). ACM, New York, NY, USA, 78–83. https://doi.org/10.1145/3336294.3336322

[33] 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, Piscataway, NJ, USA, 240–251. https://doi.org/10.1109/ICST.2019.00032

[34] 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 (Jan. 2023), 28:29. https://doi.org/10.1007/s10664-022-10265-9

[35] 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* (Oct. 2023). To appear.

[36] Thomas Thüm. 2020. A BDD for Linux? The Knowledge Compilation Challenge for Variability. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Montreal, QC, Canada). ACM, New York, NY, USA, Article 16, 6 pages. https://doi.org/10.1145/3382025.3414943

[37] Thomas Thüm, Christian Kästner, Sebastian Erdweg, and Norbert Siegmund. 2011. Abstract Features in Feature Modeling. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Munich, Germany). IEEE, Washington, DC, USA, 191–200. https://doi.org/10.1109/SPLC.2011.53

[38] Marc Thurley. 2006. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, Berlin, Heidelberg, Germany, 424–429.

[39] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A Classification of Product Sampling for Software Product Lines. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)* (Gothenburg, Sweden). ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3233027.3233035