Uniform and Scalable SAT-Sampling for Configurable Systems

Conference Paper · October 2020

DOI: 10.1145/3382025.3414951

CITATIONS
2

READS
125

4 authors:

Ruben Heradio
National Distance Education University
99 PUBLICATIONS **1,214** CITATIONS
SEE PROFILE

David Fernandez-Amoros
National Distance Education University
43 PUBLICATIONS **337** CITATIONS
SEE PROFILE

José A. Galindo
Universidad de Sevilla
70 PUBLICATIONS **768** CITATIONS
SEE PROFILE

David Benavides
Universidad de Sevilla
129 PUBLICATIONS **4,373** CITATIONS
SEE PROFILE

Uniform and Scalable SAT-Sampling for Configurable Systems

Ruben Heradio[*], David Fernandez-Amoros[†], José A. Galindo[‡], David Benavides[§]

October 19, 2020

Abstract

Several relevant analyses on configurable software systems remain intractable because they require examining vast and highly-constrained configuration spaces. Those analyses could be addressed through statistical inference, i.e., working with a much more tractable sample that later supports generalizing the results obtained to the entire configuration space. To make this possible, the laws of statistical inference impose an indispensable requirement: each member of the population must be equally likely to be included in the sample, i.e., the sampling process needs to be "uniform". Various SAT-samplers have been developed for generating uniform random samples at a reasonable computational cost. Unfortunately, there is a lack of experimental validation over large configuration models to show whether the samplers indeed produce genuine uniform samples or not. This paper (i) presents a new statistical test to verify to what extent samplers accomplish uniformity and (ii) reports the evaluation of four state-of-the-art samplers: Spur, QuickSampler, Unigen2, and Smarch. According to our experimental results, only Spur satisfies both scalability and uniformity.

1 Introduction

Generating SAT solutions at random is a problem of critical importance in a variety of domains: software product line analysis and configuration [42, 30, 37], software testing [9, 16, 48, 47], integrated circuit simulation and verification [54, 40, 23], etc.

To grasp this problem's relevancy and complexity, let us present an example taken from the software product lines domain. *BusyBox*[1] is a software tool that replaces many standard GNU/Linux utilities with a single small executable, thus providing an environment customized for a diversity of embedded systems. To achieve size-optimization, BusyBox is remarkably modular, supporting the inclusion/exclusion of 613 features at compile time. These features and their interrelationships are specified with a configuration language named *Kconfig*[2]. To guarantee that every valid configuration satisfies all dependencies, the Kconfig model is translated into a

[*]rheradio@issi.uned.es
[†]david@issi.uned.es
[‡]jagalindo@us.es
[§]benavides@us.es
[1]https://busybox.net/
[2]https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt

Boolean formula that is then processed with a logic engine [6, 18] (e.g., a SAT solver [8]). Therefore, a valid configuration corresponds to a *satisfiable assignment* of the formula, also called, a *SAT solution* [47] or a *witness* [9].

As a consequence of the inter-feature dependencies, the space of valid configurations ($7.428 \cdot 10^{146}$) is a tiny portion of the whole configuration space (2^{613}): only $2.185 \cdot 10^{-36}\%$ of the possible configurations are valid[3]. Nevertheless, the *population* of valid configurations is still huge, and consequently, many relevant issues cannot be handled by examining every valid configuration.

For instance, Halin et al. [21] adopted an exhaustive strategy to test the *JHipster*[4] system, checking all its valid configurations. JHipster is a code generator for web applications with only 45 selectable features that can produce a total of 26,256 valid configurations. Checking this modest configuration space with the *INRIA Grid'5000*[5] required 4,376 hours of CPU time (~ 182 days), and 5.2 terabytes of disk space.

Recently, several researchers have advocated approaching this and other issues related to configurable systems through statistical inference [50, 38, 53, 20, 26, 29, 2, 5]; that is, working with a much more tractable *sample* that later supports generalizing the results obtained to the population. An essential requirement that any sample must satisfy to be genuinely representative of the population is that it is collected at random [27]. In other words, each member of the population must be *equally likely* to be included in the sample. Authors often use the term *uniform* random sampling [47, 49] to emphasize this idea.

A naive approach to get such a sample would be (i) generating a random configuration set without considering the feature dependencies, and then (ii) checking with a logic engine if the configurations conform to the dependencies. Unfortunately, and as mentioned above, feature dependencies shrink the configuration space extraordinarily, and so getting a valid configuration at pure random is extremely unlikely. As a result, more elaborated algorithms have been proposed to generate uniform random samples at a reasonable computational cost. Moreover, verifying that these algorithms and their actual implementations indeed generate genuine uniform samples is a challenge by itself, and thus distinct procedures have been proposed to certify sampler uniformity.

A severe shortcoming of available uniformity certification procedures is that they require gigantic sample sizes to produce reliable results [16, 1, 9]. As a consequence, sampler uniformity has been checked only on miniature models so far, which hinders the external validity of the results. This paper presents a new statistical test that overcomes this limitation by reducing the sample size enormously ($\sim 99\%$). The paper also reports the empirical evaluation of the scalability and uniformity of four state-of-the-art samplers on configuration models with up to 18,570 features: *Spur* [1], *QuickSampler* [16], *Unigen2* [11, 10], and *Smarch* [43]. The obtained results reveal that only Spur satisfies both scalability and uniformity.

The remaining of this paper is organized as follows: Section 2 overviews prior research on uniform random samplers and techniques to assess to what extent samplers accomplish uniformity. Section 3 details our goodness-of-fit test. Section 4 applies our test to evaluate Spur, QuickSampler, Unigen2, and Smarch on a benchmark composed of 218 real models. Finally, Section 5 summarizes the main conclusions of our work.

[3] https://github.com/rheradio/VMStatAnal

[4] https://www.jhipster.tech/

[5] https://www.grid5000.fr/

2 Uniform random sampling: state-of-the-art

As an illustrative example of the relevance that SAT solution sampling has to approach problems related to configurable software systems, in the 23rd International Systems and Software Product Line Conference (SPLC), there was a *challenge* dedicated specifically to this topic and entitled "Product Sampling for Product Lines: The Scalability Challenge" [46]. Moreover, different papers have been recently published on uniform random sampling, and other sorts of sampling such as t-wise, in SPLC [52, 44, 37] and the International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS) [31].

2.1 Uniform random samplers

The following sections summarize the foremost strategies to generate uniform random samples for a model encoded as a Boolean formula φ. It is worth remarking that this paper adopts the sampling terminology typically used in statistics (e.g., see [12, 27]): a *sample* is a collection of *cases*. In this paper context, a case is a SAT solution of φ. The *sample size* is the number of cases in the sample.

2.1.1 Atomic mutations

QuickSampler[6] [16] proposes a heuristic procedure to gain scalability by minimizing the number of calls to a constraint solver. It starts by generating a random variable assignment without taking into account the formula constraints. Logically, the assignment often violates some constraints and thus it is unsatisfiable. So, QuickSampler calls the Z3 solver [15] to fix the assignment by providing a MAX-SAT solution. Then, QuickSampler flips the value of each variable and calls again Z3 to get the correspondent SAT assignment. The differences between the variable values of the original and flipped SAT assignments are called *atomic mutations*. By combining those mutations, QuickSampler quickly generates many new assignments without calling the solver because, according to the experiments reported by QuickSampler's authors, those combinations usually happen to be satisfiable.

2.1.2 Hashing-based sampling

Several techniques divide the space of SAT solutions into small "cells" of approximately the same size using r independent hash functions. Accordingly, sampling is done by choosing a cell at random, and then generating a satisfying assignment for that cell using a SAT solver. A critical point of these techniques is determining the "right" r value. For instance, Bellare et al. [7] showed that an r equal to the number of formula variables guarantees uniformity. However, Chakraborty et al. [11] reported that such r does not scale in practice; in contrast, $r = 3$ scales better and ensures near-uniformity. *Unigen2*[7] [10] develops these ideas further, giving stronger uniformity guarantees.

2.1.3 Counting-based sampling

In Section 7.1.4 of [28], Knuth showed how to accomplish uniform random sampling by subsequently partitioning the SAT solution space on variable assignments, and then counting the number of solutions of the resulting parts.

[6] https://github.com/RafaelTupynamba/quicksampler
[7] https://bitbucket.org/kuldeepmeel/unigen

Conceptually, the procedure works as follows: first, the number of solutions #SAT(φ) of the input formula φ with v variables $x_1, x_2, \ldots, x_v$ is computed. Then, the number of solutions where x_1 is true is counted : #SAT$(\varphi \wedge x_1)$. Therefore, x_1 follows a Bernoulli distribution with probability $p_1 = \frac{\#\mathrm{SAT}(\varphi \wedge x_1)}{\#\mathrm{SAT}(\varphi)}$, and accordingly, its random assignment is generated. For instance, imagine that x_1 is assigned to false. Then, x_2 would follow a Bernoulli distribution with probability $p_2 = \frac{\#\mathrm{SAT}(\varphi \wedge \bar{x}_1 \wedge x_2)}{\#\mathrm{SAT}(\varphi \wedge \bar{x}_1)}$, and it would be randomly assigned. The procedure advances until the last variable x_v is assigned, and thus the random solution is completed.

The original algorithm by Knuth is specified on BDDs very efficiently, as the probabilities required for all the possible SAT solutions are computed just once with a single BDD traversal, and then reused every time a solution is generated. Nevertheless, Knuth's algorithm can be easily adapted to SAT technology. In particular, *Spur*[8] [1] and *Smarch*[9] [43] rely on a *#SAT-solver* named *sharpSAT* [51].

2.2 Methods for testing sample uniformity

The following methods have been devised to test the uniformity of a random sampler $\mathbb{S}$:

2.2.1 Method 1: Generate a massive sample with $\mathbb{S}$, and compare it with another one obtained simulating an ideal uniform sampler

This is the most common technique in the literature [1, 16, 47, 10, 49]. First, the total number n of SAT solutions is counted for the Boolean formula φ, typically using a #SAT-solver. Having n, the generation of a uniform sample with size s is simulated as follows: imagine that numbers $1, 2, \ldots, n$ are put into a box; then, s numbers are sampled with replacement from the box, guaranteeing that the probability each number has to be extracted is $\frac{1}{n}$.

For example, JHipster encompasses 26,256 valid configurations [21]. Figure 1 shows the histogram of a sample ten times greater than the number of configurations, which has been obtained sampling with replacement from the set $\{1, 2, \ldots, 26256\}$. The x-axis depicts numbers' occurrences, i.e., there are numbers that appear $0, 1, \ldots, 27$ times in the sample; the y-axis shows how frequent are those occurrences in the sample. As expected, most numbers appear ten times (see the red vertical line in Figure 1), however, and due to randomness, some numbers appear more frequently than others.

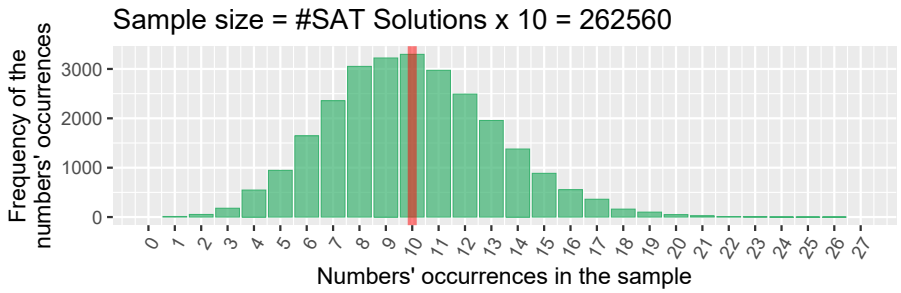

Figure 1: Simulated uniform random sample of the JHipster configuration model

[8] https://github.com/ZaydH/spur
[9] https://github.com/jeho-oh/Kclause_Smarch

Another sample with size s, in this case composed of SAT solutions, is then generated with the sampler $\mathbb{S}$. For this sample, a counterpart histogram to Figure 1 is obtained, representing how often solutions appear in the sample.

Finally, the uniformity of $\mathbb{S}$ is verified by measuring the distance between both histograms, using, for instance, the Kullback-Leibler divergence [1].

Unfortunately, this method has a severe limitation: it does not scale except for formulas with a reduced number of SAT-solutions because, to produce reliable results, s needs to be much larger than n (see [1, 16] for an explanation). For example, Dutra et al. [16] propose $s \geq 5n$. As the number of solutions grows exponentially with the number of variables of φ, the method only works for the simplest models with just a few features.

2.2.2 Method 2: Assume the existence of a uniform sampler $\mathbb{U}$, and compare the samples generated by both $\mathbb{S}$ and $\mathbb{U}$

.

Chakraborty and Meel [9] proposed this method, providing its corresponding implementation as well, called *barbarik*[10]. The method makes a strong supposition: there is available a sampler $\mathbb{U}$ that is known to be uniform. Thus two samples of the same size s are generated with $\mathbb{S}$ and $\mathbb{U}$ and, depending on the distance between the samples, barbarik decides if $\mathbb{S}$ is approximately uniform.

The key of the method is how to define "approximately" for reaching a balance between uniformity and sample size, i.e., for avoiding the large s that Method 1 requires. Two parameters, called tolerance ϵ and intolerance η, support adjusting the uniformity criterion. A sampler is uniform whenever the probability $p_1, p_2, \ldots, p_n$ of all the n SAT solutions is exactly $\frac{1}{n}$.

Barbarik relaxes this definition, proposing that a sampler is *additive almost-uniform* if $p_1, p_2, \ldots, p_n \in \left[\frac{1-\epsilon}{n}, \frac{1+\epsilon}{n}\right]$. Moreover, a sampler is η*-far* from uniformity if $\left|\sum_{i=1}^{n} p_i - \frac{1}{n}\right| \geq \eta$.

Chakraborty and Meel claim that s depends on ϵ and η exclusively, but not on n. In particular, they state that a uniformity test with *significance level* $\alpha = 0.1$ (i.e., 0.9 probability of accepting the uniformity of a sampler when it is genuinely uniform) and *power* $\beta = 0.1$ (i.e., 0.9 probability of rejecting the uniformity of a sampler that indeed it is not uniform) is accomplished when $\epsilon = 0.6$ and $\eta = 0.9$, requiring a sample size of $1,729,750$. Unfortunately, they do not provide a detailed formal proof for these questions in [9].

An evident weakness of this method is the necessity of a sampler $\mathbb{U}$ with certified uniformity as a support lever. It is worth noting that, although an algorithm can be proven to generate uniform samples theoretically, some of its implementations may have errors. In other words, every sampling program needs to be tested, and thus Method 2 implicitly assumes the existence of another reliable uniformity testing method.

2.2.3 Method 3: Compare the theoretical variable probabilities in φ with the empirical variable frequencies in a sample generated with $\mathbb{S}$

Plazar et al.'s method [47] begins computing the theoretical probability each variable x has to appear in a SAT solution. To do so, the procedure introduced in Section 2.1.3 is adopted, calling a #SAT solver repeatedly, one time per variable. #SAT(φ) gives the total number of SAT solutions, and #SAT$(\varphi \wedge x)$ calculates the number of solutions where x is true. Hence, the probability of x is $p = \frac{\#\mathrm{SAT}(\varphi \wedge x)}{\#\mathrm{SAT}(\varphi)}$. Likewise, if x is true t times in a sample

[10] https://github.com/meelgroup/barbarik

of size s, its empirical frequency is $f = \frac{t}{s}$. Then, the *deviation* between p and f is $d = 100 \cdot \frac{|p-f|}{p}$. Finally, Plazar et al. propose two *thresholds* for d: (i) when $d \leq 10$ for all variables, the deviations are *very low*, and thus sampler uniformity is accepted; (ii) when $d \geq 50$ for some variables, they show *very high* deviations, and so uniformity is rejected. Regarding the sample size, Plazar et al. propose always using $s \sim 10^6$, independently of the number of variables of φ (no formal justification is given for this specific value in [47]).

Regrettably, this method often throws false negatives for formulas where many variables have low probabilities. For instance, let us suppose that, for some variable, $p = 0.01$. Then, a genuine uniform sampler might easily generate a sample where f is slightly different just due to randomness, e.g., $f = 0.015$. Therefore, $d = 100 \cdot \frac{|0.01-0.015|}{0.01} = 50$, and thus the sampler uniformity would be rejected. The chances that these types of wrong diagnoses happen increases with the number of low-probability variables, and it is worth noting that real models with numerous low-probability variables are not "corner cases"; for example, in three out of the seven configuration models analyzed in [22], more than 46% of their variables have $p \leq 0.05$: the open-source project *Fiasco v2014092821*, the *Dell* laptop configurator, and the *Automotive 02* system.

A major shortcoming of all existing uniformity testing methods is the large sample size they need. For instance, in [1] and [49], Method 1 is applied on a model called `blasted_case110` with 287 variables, requiring $s = 4 \cdot 10^6$ SAT solutions. In [9], Method 2 is used on `blasted_case110` as well, needing this time $1,729,750$ SAT solutions to ensure probability errors of Type 1 $\alpha = 0.1$ and Type 2 $\beta = 0.1$. As we will see in Section 4, our method provides stronger test guarantees ($\alpha = 0.01$ and $\beta = 0.01$) for `blasted_case110` with a minimal sample size of 13,027 solutions (i.e., a 99.25% sample size reduction with respect to Method 2).

Summary

Three methods have been proposed to verify samplers' uniformity. Methods 1 and 3 only provide a numerical distance. Method 2 gives the statistical distribution of the distance as well, thus supporting the estimation of Type I and II errors.
Unfortunately, all methods require a huge sample size, thus uniformity has been tested so far on trivial models, with less than 100 features.

3 Assessing the uniformity of SAT solution samplers

Figure 2 sketches the method we propose for verifying if a sampler generates uniform random samples of a model encoded as a Boolean formula φ. Similarly to *Method 3* discussed in Section 2.2.3, our approach compares *empirical information* about a sample with *theoretical information* about the whole population of SAT solutions that the model represents. Nevertheless, instead of using the limited Method 3 *deviation* measure, we use a statistical *goodness-of-test* that (i) has a robust mathematical basis, (ii) estimates the *statistical significance* of the results (i.e., how generalizable they are), and (iii) supports adjusting the sample size according to precise statistical criteria (i.e., *Type I* and *II errors*, and *effect size*). First, Section 3.1 explains this statistical goodness-of-fit test. Afterwards, Section 3.2 discusses how to determine a convenient sample size for the goodness-of-fit test.

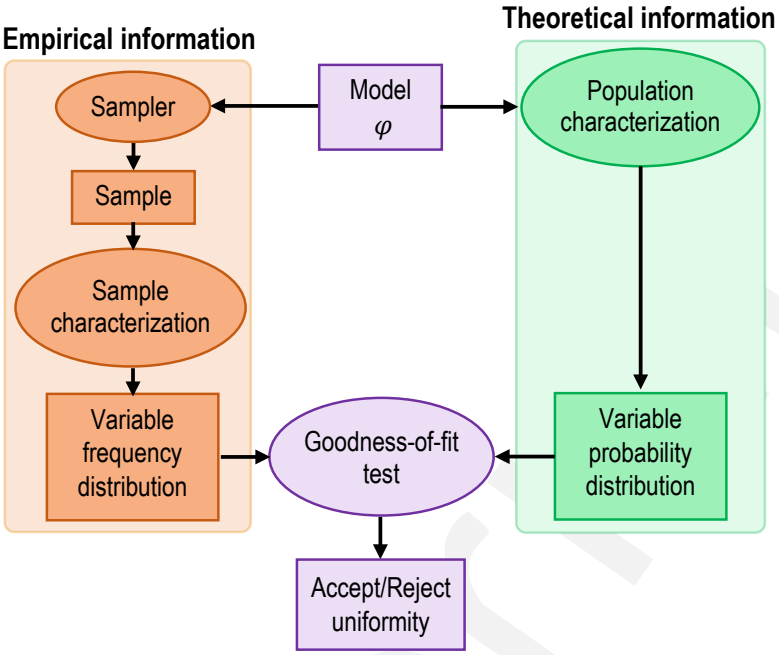

Figure 2: Proposed method for verifying if a sampler generates uniform samples of a model φ

3.1 Goodness-of-fit testing

In statistics, the methods for examining how well a sample agrees with the population distribution are known as *goodness-of-fit* testing [14]. Those methods require characterizing both the sample and the population in terms of a quantitative measure. In particular, our characterization will be based on the variable probability distribution.

Computing the distribution of the variable frequencies in a sample is straightforward: traverse the sample and calculate, for each variable, the proportion of cases where the variable is true. Nevertheless, obtaining the variable probabilities for the whole population is more complicated. In Section 2.2.3, we saw that those probabilities can be computed by calling a #SAT solver repeatedly; in particular, if n is the number of variables of φ, the #SAT solver needs to be called $n+1$ times, i.e., once to get #SAT(φ), and n times to compute #SAT$(\varphi \wedge x_i)$. It is worth noting that the #SAT problem is computationally even more expensive than the SAT one [8], and so #SAT-solver calls often take considerably more time than SAT-solver calls.

Fortunately, much more efficient methods can often be applied to calculate variable probabilities in large configuration models. For instance, (i) if φ has n variables and it is encoded as a *Binary Decision Diagram* (BDD) with v nodes, Heradio et al.'s algorithm [22] computes the probabilities in $O(nv)$ time complexity[11]; or (ii) if the configuration model is specified as a *feature model* with n features, m of which appear in *crosstree-constraints*, Fernandez-Amoros et al.'s algorithm [17] does it in $O(n^2 2^m)$.

Once the probabilities are known, the goodness-of-fit can be tested graphically. For example, the density

[11]The main weakness of BDDs is finding good variable and clause orders that minimize the computer memory required for the BDD. Although finding optimal orders is an NP-problem, there are several strategies to handle this issue heuristically [35, 3, 4, 39, 36].

plot in Figure 3 provides an exploratory goodness-of-fit analysis of four samples generated with Spur, Quicksampler, Unigen2, and Smarch for the JHipster configuration model; every sample is composed of 5,994 SAT solutions (Section 3.2 provides a justification for this specific sample size). Figure 3 represents the distribution of the absolute differences between every pair f_i and p_i corresponding to the frequency and probability of the variable x_i in the sample and the population, respectively. The samples generated with Spur, Unigen2, and Smarch match the population distribution adequately, as all differences are close to zero. In contrast, QuickSampler's sample departs considerably from the population distribution because it produces large differences, some of them even greater than 0.5.

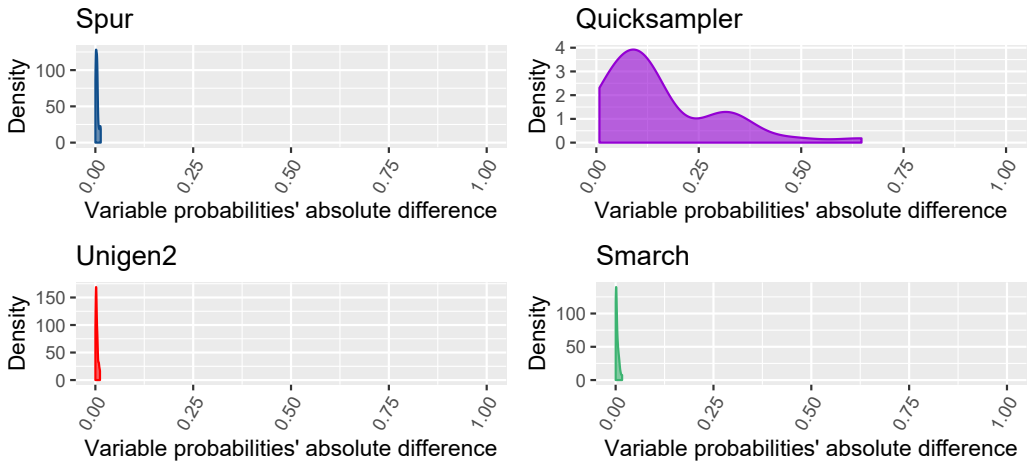

Figure 3: Density plot showing the distribution of the absolute difference between the population and sample variable probabilities for the JHipster configuration model

After exploring the samples goodness-of-fit graphically, it is desirable to advance towards a more formal test that provides an accurate numerical quantification. For that, the distance between the sample and population distributions is measured with the *Jensen-Shannon divergence* [34].

Let φ be the Boolean formula that encodes a model with n variables. To compare variable frequencies and probabilities, two vectors F and P are considered: (i) $F = [f_1, \ldots, f_n]$ stores the empirical frequencies, i.e., f_i is how many SAT solutions in the sample have variable x_i assigned to true, divided by the sample size; (ii) $P = [p_1, \ldots, p_n]$ stores the variable probabilities.

To avoid worthless comparisons, i-elements with $p_i = 0$ or $p_i = 1$ are removed from F and P. In SPL terminology, these x_i variables are called *dead* and *core features*, respectively [45]. As all solutions in the sample are guaranteed to be valid, the empirical frequencies of dead and core features are necessarily 0 and 1 as well.

Then, the Jensen-Shannon divergence between F and P, denoted $D(F, P)$, is calculated as:

$$D(F,P) = \sum_{\forall i | 0<p_i<1} f_i \log_2 \frac{f_i}{\frac{f_i}{2} + \frac{p_i}{2}} + \sum_{\forall i | 0<p_i<1} p_i \log_2 \frac{p_i}{\frac{p_i}{2} + \frac{f_i}{2}} \quad (1)$$

$D(F, P)$ measures to what extent the difference between F and P is greater than expected by chance if F corresponded to a uniform random sample. In the extreme cases, $D(F, P) = 0$ when F totally matches P, and $D(F, P) = 1$ when the F completely disagrees with P.

Nevertheless, D is a mere distance, i.e., we cannot tell if D is *significantly* greater than expected due to randomness. Therefore, a *statistical inference test* is needed to quantify how generalizable the obtained distance is, i.e., a test that estimates the probability of a specific value of $D(F, P)$ assuming that the sampler is genuinely uniform. In the case that the estimated probability is excessively low, it is unlikely that the disagreement between F and P is due to chance, and so we can conclude that the sampler is not uniform.

Let s be the sample size, and m the number of elements in P that are neither zero nor one (e.g., the JHipster model has 45 features, there are seven core features and no dead features, so $m = 38$). According to the proof given by Grosse et al. in Section 4.C of [19], $2s(\ln 2)D(F, P)$ has a χ^2 distribution with $m-1$ degrees of freedom. As a result, a *Chi-Squared goodness-of-fit test* [14] built upon the statistic $2s(\ln 2)D(F, P)$ will help us to decide whether the sampler is uniform.

Summary

Samplers' uniformity can be checked with a standard Chi-Squared goodness-of-fit test built upon the Jensen-Shannon divergence of the variable probabilities.

3.2 Sample size

In contrast to typical *Null Hypothesis Significance Tests* (NHSTs), where the null hypothesis H_0 states the opposite to what the researcher pursues to demonstrate, goodness-of-fit tests are a special case of NHSTs [14] where H_0 is: "the sample agrees with the population". Accordingly, the reliability of these tests depends on the following parameters:

- The *significance level* α sets the probability of making a *Type I error*, i.e., the probability of rejecting H_0 when it is indeed true. It is worth noting that α is also the threshold for rejecting H_0 (i.e., H_0 is rejected whenever p-value $< \alpha$).
- β sets the probability of making a *Type II error*, i.e., the probability of accepting a false H_0.

When H_0 is false, *it is false to some degree*. That degree is measured by another parameter called *effect size* [33]. In particular, Cohen [13] proposes the index w for measuring the effect size in Chi-Squared tests. As a rule of thumb, w values around 0.1, 0.3, and 0.5 correspond to *small*, *medium*, and *large* effect sizes, respectively.

Interestingly, sample size, effect size, α, and β have an intimate relationship in NHSTs: given any three of them, the fourth can be determined. In Section 7.3 of [13], Cohen provides different *power tables* to calculate the minimum sample size required to ensure the reliability of a Chi-Squared test given the values of α, β, w, and χ^2's degrees of freedom. Nowadays, many statistical packages provide those tables, e.g., see Chapter 10 of [25].

In Section 3.1 we saw that the goodness-of-fit of any sample from the JHipster configuration model can be undertaken with a Chi-Squared test with $m - 1 = 37$ degrees of freedom. Then, according to Cohen's power tables, the required sample size is 5,994 cases when $\alpha = 0.01, \beta = 0.01$, and $w = 0.1$.

Summary

The sample size depends on: (i) the degrees of freedom of the Chi-Squared test, and (ii) three parameters that accommodate the reliability of the test: α, β, and w.

4 Empirical evaluation of samplers' scalability and uniformity

This paper's primary goal is to determine if there is an available sampler that fulfills both scalability and uniformity. To that end, four state-of-the-art samplers were tested experimentally:

1. *Spur*[12] [1].
2. *QuickSampler*[13] [16].
3. *Unigen2*[14] [11, 10].
4. *Smarch*[15] [43].

4.1 Research questions

Specifically, our experimental validation targets two research questions:

- **RQ1: Scalability and running time**. *Are Spur, QuickSampler, Unigen2, or Smarch able to generate samples out of any size models with a moderate running time?*
- **RQ2: Uniformity**. *Do Spur, QuickSampler, Unigen2, or Smarch generate uniform SAT solutions?*

4.2 Experimental setup

The samplers were tested against a benchmark composed of 218 models encoded as Boolean formulas in *Conjunctive Normal Form* (CNF). A CNF formula is a conjunction of *clauses*, where each clause is a disjunction of *literals* [8]. The scatter plot in Figure 4 describes the models in terms of the number of variables and clauses (the points that represent the largest models have been labelled).

Each model's sample size was determined with the procedure described in Section 3.2. In particular, the R package `pwr`[16] [25] was used to perform Cohen's power tables calculations. To ensure the highest reliability of the results, we set $\alpha = 0.01, \beta = 0.01$, and $w = 0.1$. That is, the χ^2 test confidence level was fixed to 99%, the power to 99%, and the effect size to *small*.

The histogram in Figure 5 describes the distribution of the obtained sample sizes. It is worth noting that, as Section 2.2 discusses, prior methods for testing samplers' uniformity fail to test non-trivial models because

[12] https://github.com/ZaydH/spur
[13] https://github.com/RafaelTupynamba/quicksampler
[14] https://bitbucket.org/kuldeepmeel/unigen
[15] https://github.com/jeho-oh/Kclause_Smarch
[16] https://cran.r-project.org/web/packages/pwr

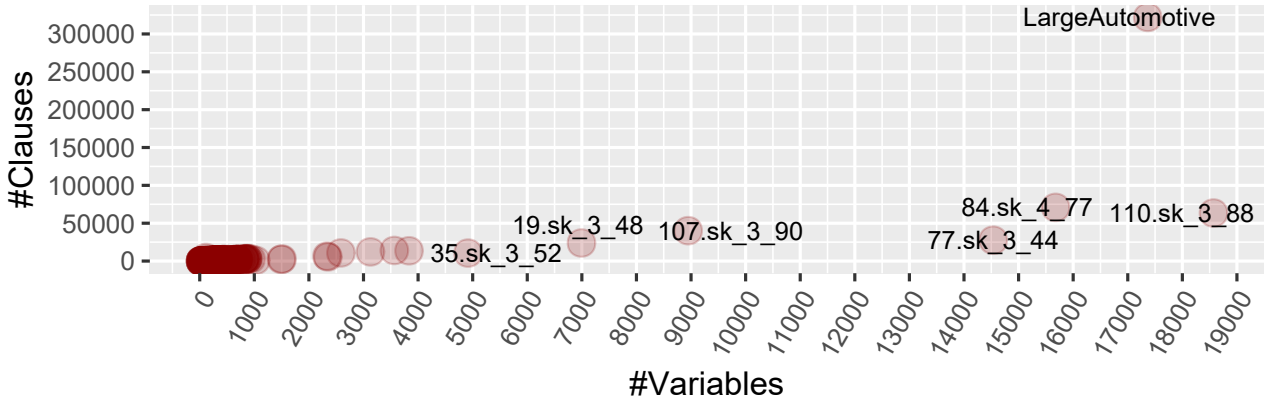

Figure 4: Size, in terms of the number of variables and clauses, of the benchmark models.

they need sample sizes greater than 10^6, even for models with less than 100 variables. In contrast, the sample size that our method requires ranges from 4,002 for the smallest model to 85,422 configurations for the biggest one.

Figure 5: Histogram of the sample sizes for $\alpha = \beta = 0.01$ and $w = 0.1$

Nine of the 218 models represent configurable software systems. The remaining models are industrial SAT formulas (mostly modeling integrated circuits) that are typically used as a benchmark in the SAT-sampling literature [10, 1, 47]. Table 1 describes the nine configuration models (the largest model is usually known as `Automotive02`, e.g., see [32]).

The samplers were run on an Intel(R) Core(TM) i7-6700HQ, 2.60GHz, 16GB RAM, operating Linux Ubuntu 19.10. They were executed on a single thread (i.e., with no parallelization), and without considering any Boolean formulas' preprocessing, such as formula's *minimal independent support* [24].

Variable probabilities were computed with Heradio et al.'s algorithm[17] [22]. The histogram in Figure 6 shows the time it took to compute the probabilities for all models in the benchmark (less than a minute for 89.45% of them). The model that needed the longest time was `blasted_case_0_b12_2`, which is an industrial SAT formula not included in Table 1. This illustrates the dependency that BDDs have on variable/clause order heuristics.

[17] https://github.com/rheradio/VMStatAnal

Model	#Vars.	#Clauses	#Cases per sample
JHipster [21]	45	104	5994
axTLS 1.5.3 http://axtls.sourceforge.net/	64	96	7198
Fiasco 2014092821 https://os.inf.tu-dresden.de/fiasco/	113	4717	7646
DellSPLOT [41]	118	2181	9131
uClibc 201 50420 https://www.uclibc.org/	298	903	13047
ToyBox 0.5.2 http://landley.net/toybox/	544	1020	10739
BusyBox 1.23.2 https://busybox.net/	613	530	18041
EmbToolkit 1.7.0 https://www.embtoolkit.org/	2331	6437	28866
LargeAutomotive [32]	17365	321897	84522

Table 1: Software configuration models included in the benchmark.

Whereas this model has a medium-size CNF formula (827 variables and 2,725 clauses), the BDD we synthesized was huge (2,644,383 nodes). Nevertheless, its computation time was still reasonable: 8.29 minutes.

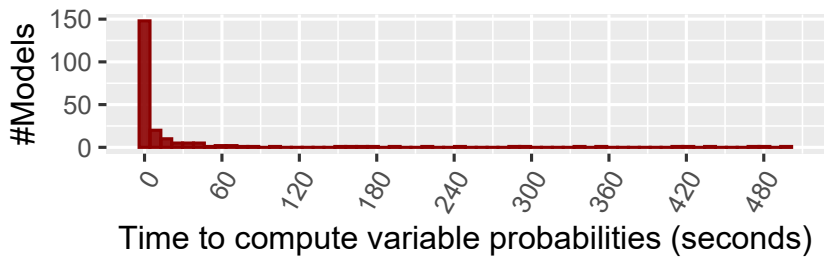

Figure 6: Histogram of the time it took to compute variable probabilities for all models in the benchmark.

4.3 Results

4.3.1 RQ1: Scalability and running time

Once each model's sample size was calculated, the samplers were run to generate the corresponding samples. The timeout was set to one hour for all generations. In total, 335.6 hours (14 days) of CPU time were needed for generating the samples (or reaching the timeout).

The scatter plot in Figure 7 represents the generation times for all samplers and models. The absence of a point in the plot reflects the sampler's inability to produce the corresponding sample. Figure 8 shows the percentage of samples that each sampler was able to generate within an hour. Table 2 provides a detailed summary of the running times for the configuration models.

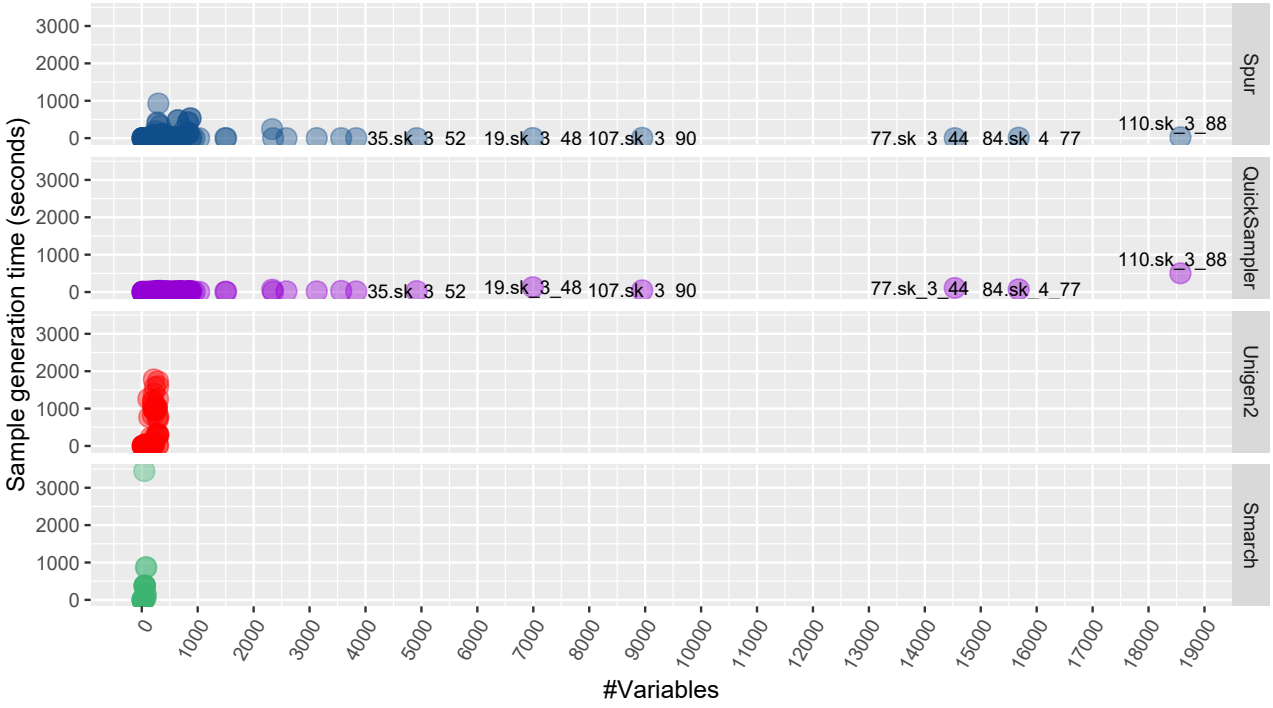

Figure 7: Sample generation time in seconds for whole benchmark (timeout = 1 hour).

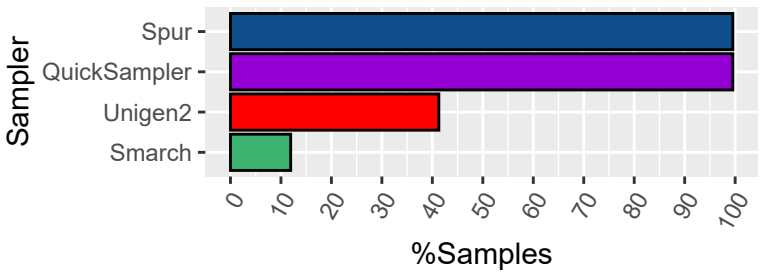

Figure 8: Percentage of samples that each sampler was able to generate.

Model	Spur	Quick Sampler	Unigen2	Smarch
JHipster	0.03	0.598	2.142	3445
axTLS	0.109	1.165	*timeout*	*timeout*
Fiasco	0.075	8.425	*timeout*	*timeout*
DellSPLOT	0.218	2.455	1258	*timeout*
uClibc	1.025	3.861	*timeout*	*timeout*
ToyBox	3.359	4.189	*timeout*	*timeout*
BusyBox	11.83	8.858	*timeout*	*timeout*
EmbToolkit	242.4	64.71	*timeout*	*timeout*
LargeAutomotive	*timeout*	*timeout*	*timeout*	*timeout*

Table 2: Sample generation time in seconds for the configuration models (timeout = 1 hour).

As α, β, and w were the same for the whole benchmark, each model's sample size depended exclusively on χ^2 degrees of freedom, i.e., on the number of variables whose probability is neither zero nor one. For instance, `110.sk_3_88` is the model with the greatest amount of variables with 18,570. However, it has an unusual number of dead features: 14,358. So its sample size was just 14,107 SAT solutions. In contrast, `LargeAutomotive` has fewer variables, 17,365, but only 6 of them are dead and 1,686 core. Hence, it needed the largest sample size of the whole benchmark: 84,522 solutions. Another large configuration model in the benchmark is `EmbToolkit`, with 2,331 variables, 619 dead features, 59 core features, and a sample size of 28,866 solutions.

According to the results, Smarch and Unigen2 had severe scalability limitations. The most complex model Smarch could tackle was `blasted_case17`, with 77 variables and 235 clauses; and the most difficult one for Unigen2 was `s510_3_2`, with 298 variables and 768 clauses.

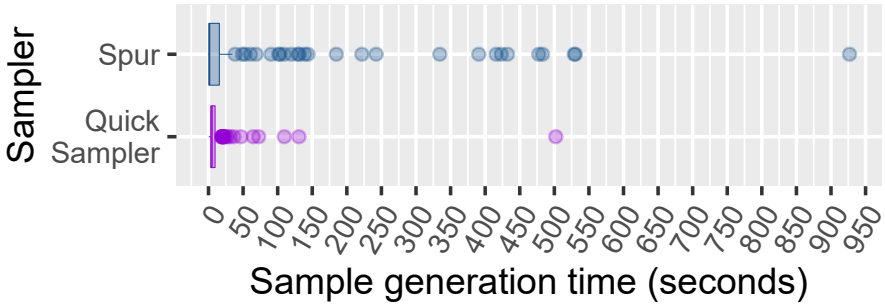

Figure 9: Box-plot of Spur and QuickSampler's sample generation times.

Spur and QuickSampler could generate the samples for all models, except for `LargeAutomotive`. Furthermore, as the box-plot in Figure 9 shows, Spur and QuickSampler produced the samples remarkably fast; the first and third quartiles of Spur's sample generation time were Q1=0.499 and Q3=15.614 seconds, and QuickSampler's quartiles were Q1=3.38 and Q3=9.467.

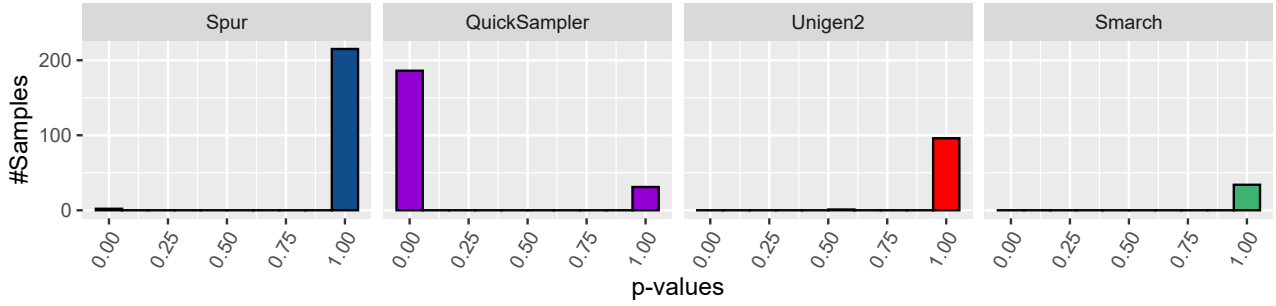

Figure 10: Goodness-of-fit p-values for the whole benchmark.

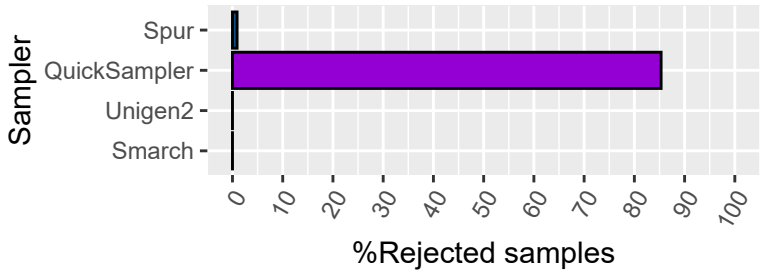

Figure 11: Percentage of rejected samples per sampler (whole benchmark)

Summary

In terms of scalability, there is a huge difference between (i) Spur and QuickSampler, which generate samples for models with thousands of variables and clauses in a few seconds, and (ii) Unigen2 and Smarch, which need more than an hour to deal with models above 300 variables and 800 clauses.

4.3.2 RQ2: Uniformity

Figure 11 shows the uniformity rejection percentage per sampler in the goodness-of-fit tests. As α was set to 0.01, the samplers' uniformity was rejected whenever the p-values fell below 0.01.

The histogram in Figure 10 provides more detailed information, as it summarizes the p-values. Spur, Unigen2, and Smarch exhibited high uniformity because their p-values were close to one in most samples. QuickSampler presented the opposite behavior, since its p-values were typically near zero.

Table 3 details the p-values for the benchmark's configuration models; N.A. stands for Not Available, and the blue and pink colors highlight the cases where the uniformity hypothesis was accepted and rejected, respectively. It is worth remarking that, since α was set to 0.01, in approximately 1% of the samples, the goodness-of-fit test made a Type I error, misjudging the sampler uniformity because an extremely low p-value happened due to randomness. For instance, this occurred in the sample that Spur generated for Embtoolkit).

Model	Spur	Quick Sampler	Unigen2	Smarch
JHipster	~ 1	~ 0	~ 1	~ 1
axTLS	~ 1	~ 0	N.A.	N.A.
Fiasco	~ 1	1.11e-16	N.A.	N.A.
DellSPLOT	~ 1	0.9462	~ 1	N.A.
uClibc	~ 1	~ 0	N.A.	N.A.
ToyBox	~ 1	~ 0	N.A.	N.A.
BusyBox	~ 1	~ 0	N.A.	N.A.
EmbToolkit	~ 0	~ 0	N.A.	N.A.
LargeAutomotive	N.A.	N.A.	N.A.	N.A.

Table 3: Goodness-of-fit p-values for the configuration models.

Summary

Spur, Unigen2, and Smarch generate uniform random samples. However, QuickSampler infringes uniformity.

4.4 Threats to validity

4.4.1 Internal validity

Our experimental design discards two potential confounders for evaluating sampler scalability: *sampling parallelization* and *use of preprocessing techniques*.

Although any sampler can be run in a multi-core fashion, thus producing samples concurrently, only Unigen2 is specifically tweaked for that. The focus of our evaluation is on the sampling techniques, not on how those techniques can be parallelized efficiently. Therefore, all samplers were run on a single thread.

There are some methods to preprocess the model Boolean formulas for speeding up further computations. For example, Ivrri et al. [24] claim that sampling with the formulas' Minimal Independent Support (MIS) produces 2-3 orders of magnitude performance improvement. Nevertheless, Plazar et al. [47] empirical results contradict that, showing no running time difference between sampling from the whole formula or the MIS. Anyway, we decided to focus on the sampling techniques, not on how any additional preprocessing methods may impact those techniques.

4.4.2 Construct validity

Regarding sampler uniformity, our experimental results display inner and outer consistency to a great extent.

Concerning inner consistency, as Figure 10 shows, the p-values that all samplers obtained for every model are remarkably similar. Spur, Unigen2, and Smarch got p-values ~ 1 in almost all models. In contrast, QuickSampler got p-values ~ 0 in 85.71% of all models, but p-values ~ 1 in the remaining ones. We found that QuickSampler's p-values ~ 1 corresponded to the smallest models, with a median number of variables and clauses equal to 51.5 and 144.5, respectivelly. That is, QuickSampler only exhibited uniformity for the most trivial models.

Regarding outer consistency, Plazar et al. [47] reported an empirical evaluation of Unigen (the previous version of Unigen2) and QuickSampler. Their results show that Unigen meets uniformity but does not scale, and QuickSampler scales but does not provide any uniformity guarantee. Our results totally agree with Plazar et al.'s findings.

4.4.3 External validity

Our study's generalizability is supported by (i) the great variety and volume of the benchmark, and (ii) the low variance in the experimental results, i.e., the consistency of the execution times and p-values throughout every model in the benchmark.

5 Conclusions

The number of SAT solutions that configuration models encompass is typically so large that most analyses cannot be performed neither examining every valid configuration, nor calling a SAT solver massively. Statistical inference opens an alternative way to address these problems by working with a much more tractable sample that later supports generalizing the results obtained to the population. However, the laws of statistical inference impose an indispensable requirement: samples must be collected at random, i.e., the configuration space needs to be covered uniformly.

We have presented a goodness-of-fit test to verify sampler uniformity, which requires a reduced sample size, even for the largest models and the most strict reliability test arrangements. As a result, we have reported the first empirical sampler uniformity evaluation on large models. According to our evaluation, there is a sampler named Spur, which generates uniform random samples and also scales for large configuration models. Hence, the power of statistical inference is ready to be unleashed for analyzing complex configurable software systems.

Material

Following *open science*'s good practices, our software artifacts are available publicly.

- The code scripts to (i) calculate each model's sample size, (ii) run the samplers, and (iii) analyze the results are available at https://github.com/rheradio/sat_sampling
- The CNF and BDD encodings of the models, together with all generated samples and statistics are available at https://doi.org/10.5281/zenodo.3757091

Acknowledgments

We would like to thank Don Batory and Jeho Oh from University of Texas at Austin (USA), and Jesús Giráldez from University of Granada (Spain) for their insightful comments in reviewing an stable version of this paper.

This work has been partially funded by the Spanish Ministry of Science, Innovation and Universities (projects VITAL-3D DPI2016-77677-P, and OPHELIA RTI2018-101204-B-C22); the Community of Madrid (research network CAM RoboCity2030 S2013/MIT-2748); the TASOVA network (MCIU-AEI TIN2017-90644-REDT); and the Junta de Andalucia (METAMORFOSIS project).

References

[1] Dimitris Achlioptas, Zayd S. Hammoudeh, and Panos Theodoropoulos. Fast sampling of perfectly uniform satisfying assignments. In *21st International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 135–147, Oxford, UK, 2018.

[2] Mauricio Alférez, Mathieu Acher, José Galindo, Benoit Baudry, and David Benavides. Modeling variability in the video domain: language and experience report. *Software Quality Journal*, 27(1):307–347, 2019.

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

[4] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation. *Journal of Universal Computer Science*, 10(12):1562–1596, 2004.

[5] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. Sampling effect on performance prediction of configurable systems: A case study. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*, page 277–288, Edmonton AB, Canada, 2020.

[6] Don S. Batory. Feature models, grammars, and propositional formulas. In *9th Software Product Line Conference (SPLC)*, pages 7–20, Rennes, France, 2005.

[7] Mihir Bellare, Oded Goldreich, and Erez Petrank. Uniform Generation of NP-Witnesses Using an NP-Oracle. *Information and Computation*, 163(2):510–526, 2000.

[8] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[9] Sourav Chakraborty and Kuldeep S. Meel. On testing of uniform samplers. In *33rd Conference on Artificial Intelligence (AAAI)*, pages 7777–7784, Honolulu, Hawaii, USA, 1 2019.

[10] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. On parallel scalable uniform SAT witness generation. In *21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 304–319, London, UK, 2015.

[11] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *25th International Conference on Computer Aided Verification (CAV)*, pages 608–623, Saint Petersburg, Russia, 2013.

[12] Laura M. Chihara and Tim C. Hesterberg. *Mathematical Statistics with Resampling and R*. Peds-R-Us Medical Education, Llc, 2011.

[13] Jacon Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 1988.

[14] Ralph B. D'Agostino. *Goodness-of-Fit-Techniques*. CRC Press, 1986.

[15] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340, Budapest, Hungary, 2008.

[16] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. Efficient Sampling of SAT Solutions for Testing. In *40th International Conference on Software Engineering (ICSE)*, pages 549–559, New York, NY, USA, 2018. ACM.

[17] David Fernandez-Amoros, Ruben Heradio, Jose A. Cerrada, and Carlos Cerrada. A Scalable Approach to Exact Model and Commonality Counting for Extended Feature Models. *IEEE Transactions on Software Engineering*, 40(9):895–910, 2014.

[18] David Fernandez-Amoros, Ruben Heradio, Christoph Mayr-Dorn, and Alexander Egyed. A Kconfig translation to logic with one-way validation system. In *23rd International Systems and Software Product Line Conference (SPLC)*, page 303–308, Paris, France, 2019.

[19] Ivo Grosse, Pedro Bernaola-Galvan, Pedro Carpena, Ramon Roman-Roldan, Jose Oliver, and H. Eugene Stanley. Analysis of symbolic sequences using the jensen-shannon divergence. *Physical Review E*, 65(2):041905/1–0419051/16, 2002.

[20] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. Data-efficient performance learning for configurable systems. *Empirical Software Engineering*, 23(3):1826–1867, 2018.

[21] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering*, 24(2):674–717, 2019.

[22] Ruben Heradio, David Fernandez-Amoros, Christoph Mayr-Dorn, and Alexander Egyed. Supporting the statistical analysis of variability models. In *41st International Conference on Software Engineering (ICSE)*, pages 843–853, Montréal, Canada, 2019.

[23] Michael Hübner and Jürgen Becker. *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*. Springer, 2011.

[24] Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, 21(1):41–58, 2016.

[25] Robert Kabacoff. *R in Action: Data analysis and graphics with R*. Manning Publications, 2011.

[26] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel. Distance-based sampling of software configuration spaces. In *41st International Conference on Software Engineering (ICSE)*, pages 1084–1094, Montreal, Canada, 2019.

[27] Daniel Kaplan. *Statistical Modeling: A Fresh Approach*. Peds-R-Us Medical Education, Llc, 2012.

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

[29] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. Tradeoffs in modeling performance of highly configurable software systems. *Software & Systems Modeling*, 18(3):2265–2283, 2019.

[30] Sebastian Krieter. Enabling Efficient Automated Configuration Generation and Management. In *23rd International Systems and Software Product Line Conference (SPLC)*, pages 215–221, Paris, France, 2019.

[31] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. Yasa: Yet another sampling algorithm. In *International Working Conference on Variability Modelling of Software-Intensive System (VaMoS)*, Magdeburg, Germany, 2020. Association for Computing Machinery.

[32] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. Propagating configuration decisions with modal implication graphs. In *40th International Conference on Software Engineering (ICSE)*, pages 898–909, Gothenburg, Sweden, May 2018.

[33] Daniel Lakens. Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs. *Frontiers in Psychology*, 4(863):1–12, 2013.

[34] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, Jan 1991.

[35] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations And Applications*, chapter Ch. 9: Optimizing the Variable Order, pages 145–170. Springer, 1998.

[36] Marcilio Mendonça. *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, 2009.

[37] Daniel-Jesus Muñoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *23rd International Systems and Software Product Line Conference (SPLC)*, pages 289–301, Paris, France, 2019.

[38] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. Using Bad Learners to Find Good Configurations. In *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 257–267, Paderborn, Germany, 2017.

[39] Nina Narodytska and Toby Walsh. Constraint and variable ordering heuristics for compiling configuration problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 149–154, Hyderabad, India, 2007.

[40] Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan Marcus, and Gil Shurek. Constraint-Based Random Stimuli Generation for Hardware Verification. In *18th Innovative Applications of Artificial Intelligence Conference (IAAI)*, pages 1720–1727, Boston, Massachusetts, USA, 2006.

[41] Alexander Nöhrer and Alexander Egyed. C2O configurator: a tool for guided decision-making. *Automated Software Engineering*, 20(2):265–296, Jun 2013.

[42] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. Finding Near-optimal Configurations in Product Lines by Random Sampling. In *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 61–71, New York, NY, USA, 2017.

[43] Jeho Oh, Paul Gazzillo, and Don Batory. t-wise Coverage by Uniform Sampling. In *23rd International Systems and Software Product Line Conference (SPLC)*, pages 84–87, New York, NY, USA, 2019. ACM.

[44] Jeho Oh, Paul Gazzillo, and Don S. Batory. *t*-wise coverage by uniform sampling. In *23rd International Systems and Software Product Line Conference (SPLC)*, pages 15:1–15:4, Paris, France, 2019.

[45] Hector Perez-Morago, Ruben Heradio, David Fernandez-Amoros, Roberto Bean, and Carlos Cerrada. Efficient Identification of Core and Dead Features in Variability Models. *IEEE Access*, 3:2333–2340, 2015.

[46] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. Product sampling for product lines: The scalability challenge. In *23rd International Systems and Software Product Line Conference (SPLC)*, page 78–83, Paris, France, 2019.

[47] Quentin Plazar, Mathieu Acher, Gilles Perrouin, Xavier Devroey, and Maxime Cordy. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet? In *12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 240–251, Xian, China, China, 2019.

[48] Subhajit Roy, Awanish Pandey, Brendan Dolan-Gavitt, and Yu Hu. Bug Synthesis: Challenging Bug-Finding Tools with Deep Faults. In *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 224–234, Lake Buena Vista, Florida, USA, 2018.

[49] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. Knowledge Compilation meets Uniform Sampling. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 620–636, Awassa, Ethiopia, 2018.

[50] Paul Temple, José A. Galindo, Mathieu Acher, and Jean-Marc Jézéquel. Using machine learning to infer constraints for product lines. In *20th International Systems and Software Product Line Conference (SPLC)*, pages 209–218, Beijing, China, 2016.

[51] Marc Thurley. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In *9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 424–429, Seattle, WA, USA, 2006.

[52] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. A classification of product sampling for software product lines. In *International Systems and Software Product Line Conference (SPLC)*, page 1–13, Gothenburg, Sweden, 2018. Association for Computing Machinery.

[53] Markus Weckesser, Roland Kluge, Martin Pfannemüller, Michael Matthé, Andy Schürr, and Christian Becker. Optimal reconfiguration of dynamic software product lines based on performance-influence models. In *22nd International Systems and Software Product Line Conference (SPLC)*, pages 98–109, Gothenburg, Sweden, 2018.

[54] Jun Yuan, Ken Albin, Adnan Aziz, and Carl Pixley. Simplifying Constraint Solving in Random Simulation Generation. In *11th IEEE/ACM International Workshop on Logic & Synthesis (IWLS)*, pages 185–190, New Orleans, Louisiana, USA, 2002.