Preprocessing is What You Need: Understanding and Predicting the Complexity of SAT-based Uniform Random Sampling

Olivier Zeyen
University of Luxembourg, SnT
Luxembourg

Maxime Cordy
University of Luxembourg, SnT
Luxembourg

Gilles Perrouin
PReCISE/NaDI, University of Namur
Belgium

Mathieu Acher
Univ Rennes, Inria, CNRS, IRISA
France

ABSTRACT

Despite its NP-completeness, the Boolean satisfiability problem gave birth to highly efficient tools that are able to find solutions to a Boolean formula and compute their number. Boolean formulae compactly encode huge, constrained search spaces for variability-intensive systems, e.g., the possible configurations of the Linux kernel. These search spaces are generally too big to explore exhaustively, leading most testing approaches to sample a few solutions before analysing them. A desirable property of such samples is *uniformity*: each solution should get the same selection probability. This property motivated the design of uniform random samplers, relying on SAT solvers and counters and achieving different tradeoffs between uniformity and scalability. Though we can observe their performance in practice, understanding the complexity these tools face and accurately predicting it is an under-explored problem. Indeed, structural metrics such as the number of variables and clauses involved in a formula poorly predict the sampling complexity. More elaborated ones, such as minimal independent support (MIS), are intractable to compute on large formulae. We provide an efficient parallel algorithm to compute a related metric, the *number of equivalence classes*, and demonstrate that this metric is highly correlated to time and memory usage of uniform random sampling and model counting tools. We explore the role of formula preprocessing on various metrics and show its positive influence on correlations. Relying on these correlations, we train an efficient classifier (F1-score 0.97) to predict whether uniformly sampling a given formula will exceed a specified budget. Our results allow us to characterise the similarities and differences between (uniform) sampling, solving and counting.

ACM Reference Format:
Olivier Zeyen, Maxime Cordy, Gilles Perrouin, and Mathieu Acher. 2024. Preprocessing is What You Need: Understanding and Predicting the Complexity of SAT-based Uniform Random Sampling. In *Formal Methods in Software Engineering (FormaliSE) (FormaliSE '24), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3644033.3644371

FormaliSE '24, April 14–15, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0589-2/24/04.
https://doi.org/10.1145/3644033.3644371

1 INTRODUCTION

Uniform Random Sampling (URS) is a family of techniques to sample from the set of solutions of a logical formula (often, a Boolean formula), such that each solution gets the same probability of being selected. URS is a problem of both theoretical and practical interest. In particular, when testing configurable systems with hundreds of options – inducing search spaces one cannot exhaustively explore – uniform sampling is interesting as we may not know where bugs are [30, 31, 37]. Other applications include deep learning verification, where inputs are drawn from an unknown distribution [5] or evolutionary algorithms where De Perthuis de Laillevault *et al.* theoretically demonstrated the relevance of repeated uniform random sampling when initializing populations [12]. Improving URS thus benefits multiple research fields: SAT solving, software testing, machine learning, etc.

When evaluating URS techniques (or *samplers*), two quality criteria matter: *uniformity* and *scalability*. Uniformity evaluates how close the distribution of the sampled solutions is to the uniform distribution. Scalability refers to the efficiency of the sampler to produce samples within a specified amount of time, even for large formulae. Previous studies [37] demonstrated the difficulty for existing samplers to satisfy both quality criteria. Despite recent improvements [41], state-of-the-art samplers still fail to scale on complex real-world formulae (representing, e.g., the Linux kernel configurations) without sacrificing uniformity.

Why some formulae are harder to sample uniformly is *a poorly understood problem*. A simple but wrong approach to determining sampling complexity is to count the number of variables and clauses of formulae. As an example, UniGen3 [41] requires 8 seconds to produce 10000 samples from the formula `blasted_case64` – 96 variables and 299 clauses – and 13.5 seconds for the same number of samples from the `JHipster` feature model – 44 variables and 104 clauses. This indicates that these simple metrics do not adequately characterize the complexity of sampling. While there exist formula metrics that correlate with the complexity of SAT *solving* – although with varying successes [3] – the characteristics that make a formula easier or harder to sample from remain unknown.

In this paper, we assess and define meaningful metrics for understanding and predicting URS difficulty (time and memory consumption). In addition to simple metrics trivially computed from the formula structure, we consider other studied metrics in the context of SAT solving (such as the minimal independent support size and the treewidth). We also provide an efficient algorithm to compute equivalence classes [9]. To evaluate the relevance of these metrics to assess sampling and solving complexity, we consider two uniform random samplers, SPUR [2] and UniGen3 [41], as well as SAT solvers [11, 14] and a model counter [25]. Motivated by previous studies showing that the formulae encoding the variability spaces of configurable systems tend to be harder to uniformly sample than others [16, 37], we built a diversified dataset of 488 SAT formulae, 128 of which are encoding configurable systems.

Equipped with a set of metrics measured on various formulae, we measure correlations between these metrics and the performance of uniform samplers. We demonstrate the existence of strong correlations (Kendall coefficients > 60) between some (combinations of) metrics and sampling complexity (time and memory consumption). We also demonstrate the positive role of *formula preprocessing*, i.e., computing independent support for the formulae and applying the metrics on them, for complexity prediction. Next, we lean on these results and develop a classification model to predict whether a given sampling problem (i.e. using a given uniform sampler to sample from a given formula) is affordable for a given time and memory budget. We evaluate our model on all our 488 subject formulae and show that it can achieve at best a classification F1-score of 0.97 and an AUC-ROC of 0.98.

To summarize, this paper makes the following contributions:

(1) ***Correlation study.*** We study the correlation between the complexity metrics and the computational cost of sampling (time and memory). We demonstrate a strong correlation between the number of equivalence classes and sampling cost.
(2) ***Prediction.*** Based on these correlations, we build classification models (random forests) that leverage the metrics to classify formulae according to sampling cost, with F1-score up to 0.97 and AUC-ROC up to 0.98. We further analyze the feature importance of these models to increase our trust in the correlation study.

Open science policy. All our experimentation infrastructure is available at the following website: https://anonymous.4open.science/r/eqv_pred-9E64. The repository contains some artifacts we used to compute the treewidth or the deficiency metrics. The repository also includes the program to compute the equivalence classes and some example scripts to generate the files in the data folder. The data folder contains the resulting CSV files of our experiments. The Python scripts were used to compute the correlations and the prediction models. The repository also contains an archive with all the formulae in the DIMACS format.

2 BACKGROUND

2.1 Boolean formulae

A Boolean formula F is defined over a set of Boolean variables $Var(F)$ and takes a Boolean value that can be true or false. A literal is either a variable $x \in Var(F)$ or its negation $\neg x$, such that if variable x is set to true then the literal x evaluates to true and the literal $\neg x$ evaluates to false. We use the notations $Var(x)$ and $Var(\neg x)$ to refer to the variable corresponding to the literal x and $\neg x$, respectively, viz. variable x.

A model m of F ($m \models F$) is a set of literals such that $\forall x \in Var(F)\ x \in m \oplus \neg x \in m$ and F evaluates to true (with $\oplus$, the binary exclusive or operator). We say that a literal l evaluates to true in a model m if and only if $l \in m$. Otherwise we say that the literal l evaluates to false in m. We define R_F as the set of models m of F such that $m \models F$ if and only if $m \in R_F$. We define $|R_F|$ as the size of the set R_F.

A formula F is in negational normal form (NNF) if the negation only appears directly in front of variables. Furthermore, it is in conjunctive normal form (CNF) if written as a conjunction of disjunction of literals ($F = \bigwedge_{A_i} \bigvee_{l \in A_i} l$). A deterministic decomposable NNF (d-DNNF) is an NNF where every conjunction is *decomposable* and every disjunction is *deterministic*. A conjunction $\bigwedge N_i$ is decomposable if for every pair (j,k) we have $Var(N_j) \cap Var(N_k) = \emptyset$. A disjunction $\bigvee N_i$ is deterministic if for every pair (j,k), $R_{(N_j \wedge N_k)} = \emptyset$.

$I \subseteq Var(F)$ of a formula F is an *independent support* if every model of F can be uniquely distinguished by using the variables in I only [8, 19]. An independent support is minimal (MIS) if removing any variable from it does not yield an independent support.

Based on the above, we define the concepts of backbone and *equivalence class*:

Definition 1 (Backbone). *The backbone B_F of a formula F is defined as the set of literals that appear in each model of the formula:*

$$B_F = \{l \mid \forall m \in R_F\ (l \in m)\}$$

The backbone contains the literals that evaluate to true in every model of the formula. If we generalize the idea of equivalence between literals and the constant true to the idea of equivalence between literals we find a notion of equivalence class:

Definition 2 (Equivalence class). *An equivalence class e is a set of literals that evaluate to the same value in every model of F.*

$$\forall l, l' \in e\ \forall m \in R_F\ ((l \in m) \Leftrightarrow (l' \in m)).$$

By this definition we find that if $\{x, y\}$ is an equivalence class then $\{\neg x, \neg y\}$ is also an equivalence class. These two equivalence classes are redundant as they represent the same result. We define two equivalence classes a and b as redundant if and only if $a = b$ or $a = \{\neg x | x \in b\}$ or $a \subseteq b$ or $b \subseteq a$. If we have $a \subseteq b$ we keep b and discard a. For the rest of the paper, without loss of generality, we only consider non-redundant equivalence classes. We define the set E_F as the set of all non-redundant equivalence classes of a formula F and $|E_F|$ as the number of equivalence classes of F. Note that we necessarily have $|E_F| \leq |Var(F)|$ because we only consider non-redundant equivalence classes.

We next define three common problems for Boolean formulae, i.e., SAT solving, model counting, and URS.

Definition 3 (SAT solving). *SAT solving is the problem of finding a model m for a given formula F.*

Definition 4 (Model counting (# SAT)). *Model counting is the problem of computing the size of R_F.*

Definition 5 (Uniform Random Sampling). *URS is the problem of sampling a model from R_F such that every $m \in R_F$ has probability $\frac{1}{|R_F|}$ of being sampled.*

Despite their intrinsic links, these three problems are very different and require dedicated solutions to be addressed.

2.2 URS, Configurable Systems and Feature Models

URS is highly relevant for quality assurance activities for configurable systems, e.g., during testing [17, 37], verification [10, 32] and performance analysis [21]. This support consists of computing a representative sample of variants to infer analysis results for other variants (based on their common features with the sample). Because these variants are too numerous to be all considered for analysis, sampling offers an adequate compromise between completeness and efficiency.

Various artifacts can drive the sampling of system variants, such as feature models [22], source code, test suites, behavioral models, etc. Feature models, however, remain the most commonly used input for sampling techniques designed for configurable systems. The main reason is that the semantics of feature models can be expressed in first-order logic [6, 39], whose set of solutions corresponds to the set of valid SPL variants. This makes feature models inherently amenable to URS.

3 OBJECTIVES AND METHODS

Our objective is to understand and predict the capability (or lack thereof) of state-of-the-art samplers to sample solutions from a Boolean formula uniformly.

3.1 Research Questions

Our first research question investigates the role of metrics in the complexity of uniform random sampling:

> **RQ1:** Which metrics of Boolean formulae correlate with URS time and memory consumption?

In addition to simple characteristics like the number of variables and clauses, we consider concepts that are intensively used in the problems of SAT solving, model counting, and URS, e.g., the size of the minimal independent support and the number of *equivalence classes*.

We aim to exploit our analysis results to develop an approach that, based on the correlated characteristics, can predict whether a formula would be too costly to uniformly sample from (i.e., would exceed a predefined time and memory budget). This would enable engineers to estimate whether it is feasible to sample solutions with uniform samplers *without* wasting computation resources on intractable problems.

> **RQ2:** Can the correlated characteristics be used to predict the affordability of URS in terms of time and memory consumption?

To answer this question, we train random forest models to classify Boolean formulae into "affordable" or "not affordable", based on different combinations of the characteristics we study.

Lastly, we study whether the intrinsic links between SAT solving, model counting and sampling translate into the same influence of formula characteristics on these three problems.

> **RQ3:** Are the characteristics of Boolean formulae correlated to the complexity of URS as they are to SAT solving and model counting?

A positive answer to this question would pave the way to improve the efficiency of URS by working on the same formula transformations that reduce the difficulty of SAT solving and model counting. A negative answer would invalidate this path and call for specific solutions to reduce the complexity characteristics that impact sampling.

3.2 Complexity metrics

We consider simple metrics that are trivially computed from the structure of a Boolean formula:

- the number of variables $\#v$
- the number of clauses $\#c$
- the number of literals $\#l$

We, furthermore, consider underlying concepts that SAT solvers, counters, and samplers have used to improve the performance of their algorithm. One such metric is *#mis*, the *the size of the Minimal Independent Support* (MIS). MIS is typically computed to improve the performance of model counters (like D4 [25] and sharpSAT [43]) that some URS tools invoke during sampling.

Unfortunately computing the MIS itself may be unaffordable for complex formulae. To this end, we propose the number of *equivalence classes* (*#eqv*). The advantage over MIS is that the computation of equivalence classes only requires a simple SAT solver. We further increase the efficiency of this computation through a parallel algorithm that we develop hereafter. Using this algorithm, we compute *#eqv* for the Linux 2013 model (50000 variables) [36] in less than 1.5 wall-clock hours while computing *#mis* times out at 24 hours. Another way of approximating the MIS is to use Arjun [42], which is significantly faster than the computation of *equivalence classes*. Unfortunately, using Arjun to compute an independent support gave us lower correlations so we decided to use MIS [19] and *#eqv*. In addition, we consider other metrics that have been studied in the context of SAT solving, viz. treewidth [33] and deficiency [34]. Treewidth (tw) is used to bound the worst-case size of the decision DNNF (D-DNNF) during solving [33]. Deficiency (δ) was proven to have intrinsic links with the worst-case time complexity of SAT solving [34]. Though computing deficiency is an NP-hard problem, it can often be approximated as the number of clauses minus the number of variables.

3.3 EQV: A parallel algorithm to compute the number of equivalence classes

In [9], the authors generalize the notion of backbone to equivalence classes and propose an algorithm to compute the equivalence classes. However, their algorithm requires to add $\frac{n(n-1)}{2}$ variables to a formula with n variables – in our dataset, n can be as high as 486193 variables. Assuming every variable requires 4 bytes of RAM to be stored, the algorithm would necessitate around 472 GB of RAM to store the additional variables. This is unaffordable and prevents us from computing *#eqv* on most of the formulae we use in our experiments.

We therefore propose an adapted algorithm that requires less storage memory and can improve efficiency via parallelization. Our algorithm can divide the computation of [9] to reduce the number of added variables and enable spreading over multiple cores. It introduces an overhead, though, as it may increase the number of intermediate solver calls. As a result, our approach would run slower on a single-core computer than [9], but brings benefits on multi-core infrastructures.

Algorithm 1 EQV(ϕ)

```
Require: $\phi$ a satisfiable Boolean formula
 1: $m \leftarrow SAT(\phi)$
 2: $e \leftarrow \{m\}$
 3: $v \leftarrow \{\{x\} | x \in m\}$
 4: $crit \leftarrow mutex$
 5: do in parallel
 6: for all $\{x, y\} \in \mathcal{P}(m)$ do
 7:    $lock(crit)$
 8:    $C \leftarrow (\exists i \in e.\{x, y\} \subseteq i) \wedge \neg(\exists i \in v.\{x, y\} \subseteq i)$
 9:    $release(crit)$
10:    if $C$ then
11:       $tmp \leftarrow SAT(\phi \wedge (x \oplus y))$
12:       $lock(crit)$
13:       if $tmp = UNSAT$ then
14:          {the SAT solver proved the equivalence of $x$ and $y$}
15:          $t \leftarrow \bigcup_{i \in v | x \in i \vee y \in i} i$
16:          $v \leftarrow \{i | i \in v \wedge x \notin i \wedge y \notin i\}$
17:          $v \leftarrow v \cup \{t\}$
18:       else
19:          {the SAT solver disproved the equivalence of $x$ and $y$}
20:          $r \leftarrow \emptyset$
21:          for all $i \in e$ do
22:             $a \leftarrow \{l | l \in i \wedge l \in tmp\}$
23:             $b \leftarrow \{l | l \in i \wedge \neg l \in tmp\}$
24:             $r \leftarrow r \cup \{a\} \cup \{b\}$
25:          end for
26:          $e \leftarrow r$
27:       end if
28:       $release(crit)$
29:    end if
30: end for
31: return v
```

Our method is depicted by Algorithm 1 with $\oplus$ being the logical exclusive or operator. The algorithm uses a *SAT* procedure which takes as input a Boolean formula and either returns *UNSAT* if the formula is not satisfiable or returns the set of literals that represents the solution found by the SAT solver. The algorithm works as follows. We start by making a first call to *SAT*. Here, we suppose that the formula is satisfiable. We then suppose that the formula has only a single solution and thus consider all literals to be one equivalence class, i.e., the set e. The set e represents the set of possible but unverified equivalence classes. We pick a possible pair out of all the equivalence classes (lines 6 to 9) that is a possible candidate for an equivalence and which has not yet been proven to be correct. We call the *SAT* solver and ask for a solution in which x and y are different to disprove their equivalence. If the result is *UNSAT*, we have a proof that x and y are equivalent in all the models and we modify v accordingly. The set v thus represents the set of the verified equivalence classes. On the other hand, if the *SAT* solver returns a solution tmp we know that there exists a model of our formula in which x and y are not equivalent and thus x and y cannot be in the same equivalence class. We can also learn from the solution tmp by looking at its difference with our first model m. If two literals x and y are supposedly equal in every model then if x is present in both m and tmp, then so should y be. In other words, every change in x from m to tmp should also happen in y. Using this information, we update e between lines 18 and 24. We observe that e contains the verified separation of equivalence classes and v contains the verified unions of equivalence classes. The set e thus allows us to avoid making unnecessary *SAT* calls if we have already found two models that disprove the equivalence of two literals.

The loop on line 6 is the for loop that may be parallelized. The critical sections of Algorithm 1 may seem very large, but the data structures e and v can be updated efficiently (especially considering that v may be implemented using the UnionFind data structure). Moreover, the *SAT* calls are done outside of a critical section and thus in parallel which should grant us a significant speedup.

4 EXPERIMENTAL SETUP

We detail below the general experimental protocol that applies to all research questions. The specific settings of each research question are detailed in Section 5.

4.1 Samplers

SPUR [2]: SPUR is built on top of sharpSAT [43], a #SAT solver. Since sharpSAT essentially walks through all the solutions of a formula to count them one might think of using that to sample from a formula which is exactly what SPUR does. SPUR being tightly integrated into sharpSAT, it can exploit the way sharpSAT walks through the solutions and can thus produce uniform samples. SPUR is also one of the few samplers that comes with theoretical guarantees regarding uniformity.

UniGen3 [41]: a hashing-based algorithm. To improve UniGen2, the authors investigated the bottlenecks of UniGen2 and made key improvements to their algorithm and to the way CryptoMiniSat handles XOR formulae, leading to better performance.

We use both SPUR and UniGen3 as these are the state of the art samplers with theoretical guarantees of uniformity.

In our study, we also would like to explore the relationship between URS and SAT solving and the relationship between URS and SAT counting. To compare URS with SAT solving, we explored the two solvers MiniSAT [14] and Z3 [11]. To compare with SAT counting, we used the two state-of-the-art model counters D4 [25] and sharpSAT [43]. Since another sampler called KUS [40] is based on D4, this should also give us insights into the complexity of KUS. We do not evaluate KUS as most of the complexity related to the sampling process is absorbed by the call to D4 as demonstrated in [40].

We added an implementation of bounded SAT solving (BSAT) using Z3. BSAT is a function BSAT(ϕ, n) defined as follows: the function recursively calls Z3 on ϕ and removes the returned model from the formula until either the formula becomes unsatisfiable or the number of iterations is greater than n. BSAT is thus a form of SAT sampler which is almost guaranteed to be very far from uniform.

4.2 #SAT preprocessing

We would like to study the influence of formula preprocessing on the complexity of URS and on the correlations with our metrics. To this end we use a preprocessor called Arjun [42]. Arjun computes an independent support I of the input formula F and removes the variables that are not in the independent support I if the projection can be done in reasonable time and space. We thus obtain a new formula F' which is the projection of F on the set of variables I. Arjun ensures that $R_{F'}$ is the projection of R_F on the independent support I and that $|R_{F'}| = |R_F|$. Thus using Arjun as a preprocessor to URS does not influence the uniformity of a sampler if the sampler is guaranteed to be uniform.

4.3 Dataset

We use well-known and publicly available models in our study, which are of various complexity and are either feature models or general Boolean formulae.

4.3.1 Feature model benchmark. Overall, we use the feature models of 128 real-world configurable systems (Linux, eCos, toybox, JHipster, etc.) with varying sizes and complexity. We first rely on 117 feature models used in [23, 24]. The majority of feature models contain between 1,221 and 1,266 features. Of these 117 models, 107 comprise between 2,968 and 4,138 cross-tree constraints, while one has 14,295 and the other nine have between 49,770 and 50,606 cross-tree constraints [23, 24]. Second, we include ten additional feature models used in [26] and not in [23, 24]; they also contain a large number of features (e.g., more than 6,000). Third, we add the JHipster feature model [17, 38] to the study, a realistic but relatively small feature model (45 variables, 26,000+ configurations). We later refer to these benchmarks as the feature model benchmarks. Once put in conjunctive normal form, these instances typically contain between 1 and 15 thousand variables and up to 340 thousand clauses. The hardest of them, modeling the Linux kernel configuration, has more than 6,000 variables, and 340,000 clauses. It is generally seen as a milestone in configurable system analysis.

4.3.2 General Boolean formulae. In addition to these feature models, we have replicated the initial experiments on industrial SAT formulae as conducted in [13]. We use these results to ensure that we are using the tools with the same configurations that were previously compared. Moreover, since these original formulae are much smaller than the feature models we use (typically a few thousand clauses), they will provide a basis of results for statistical analysis, in case a solver cannot produce enough samples on the harder formulae.

4.4 Infrastructure

The experiments regarding the computation of the equivalence classes, the MIS computation as well as the time and memory usage of the samplers were computed on an HPC containing 318 nodes each of which has 256 GB of RAM and 2 AMD Epyc ROME 7H12 CPUs running at 2.6 GHz.

To measure the memory usage of the samplers we developed a wrapper program which reads the appropriate file in the /proc folder which contains information about the virtual memory usage of the program. We asked the samplers to compute 1000 samples while using less than 64 GB of RAM and in under 5 hours.

The treewidth was computed with the tool described in [18]. The correlations were computed using the SciPy Python library. To train the predictors we used Python and the scikit-learn library [35]. We used standard parameters for random forests, viz. we set the number of trees to 100, used Gini impurity for splitting, and set the number of features to consider at each split to the square root of the total number of features.

5 RESULTS

5.1 RQ1: complexity factors

Table 1 shows the Kendall rank correlation coefficients for the SPUR and UniGen3 samplers. The coefficients have been computed on the instances on which we successfully managed to compute 1000 samples in less than 5 hours and using less than 64GB of virtual memory. This means that the table was computed on 416 formulae for SPUR and 241 formulae for UniGen3. The columns #v, #c, and #l represent respectively, the number of variables, number of clauses, and the number of literals respectively, with the number of literals being the sum of the lengths of all clauses. The time and mem columns indicate the computation time and the amount of virtual memory used by a single call to Z3 respectively. We have 2 groups in our table, the regular group where we compute the correlations over our formulae and the (+Arjun) group where we first preprocess the formula with Arjun [42] and then call SPUR or UniGen3 on the output of Arjun. Some solvers take advantage of a possible MIS declaration inside of the DIMACS files. Unfortunately, not all of the solvers take advantage of the MIS declaration. We thus removed the MIS declarations from the DIMACS files. The results with the MIS declaration are nonetheless available on our companion GitHub [4]. There are no correlations between the (+Arjun) groups and the equivalence classes because Arjun automatically removes redundant variables. The time and memory usage of Arjun is ignored (the median runtime was 0.15 seconds with the longest runtime being 17 minutes). All the p-values are lower than 10^{-3}. We computed the MIS by using the tool in [19] on both the initial formulae and the preprocessed formulae. Although Arjun [42] returns an independent support, we find that the correlations are worse. We thus decided to compute the MIS with [19].

For both SPUR and UniGen3 we observe that the most correlated metrics with the computation time or the virtual memory usage is either the size of the MIS or the number of equivalence classes. However, if we add Arjun as a preprocessing step, we observe that the correlations change between SPUR and UniGen3. SPUR (+Arjun) is highly correlated with the number of clauses and with δ while UniGen3 (+Arjun) is highly correlated with the number of variables and the size of the MIS. This difference can be explained through their respective algorithms. UniGen3 adds clauses to the formula, and the size and number of added clauses depends on the number of variables (or on the MIS if the MIS is declared in the DIMACS file). SPUR on the other hand is based on an exhaustive DPLL algorithm, which means that SPUR spends a lot of time doing boolean constraint propagation which is sensitive to the number of clauses.

	#v	#c	#l	tw	δ	#mis	#eqv	time z3	mem z3
time SPUR	45.741	48.502	50.000	34.582	49.263	**62.255**	**68.474**	34.223	39.988
mem SPUR	42.213	46.195	47.966	31.556	47.062	**60.151**	**62.761**	32.670	37.964
time SPUR (+Arjun)	58.865	**79.125**	**75.663**	**64.014**	**75.521**	50.814	-	34.552	46.589
mem SPUR (+Arjun)	**61.984**	**79.073**	**75.190**	**64.100**	**74.240**	53.920	-	35.606	47.502
time UniGen3	47.230	45.545	45.017	34.614	44.192	54.574	**74.890**	25.000	22.266
mem UniGen3	47.819	45.247	45.023	37.603	43.475	**68.683**	**71.353**	24.887	24.403
time UniGen3 (+Arjun)	**88.902**	46.035	44.769	41.602	44.005	**81.159**	-	21.965	32.522
mem UniGen3 (+Arjun)	**86.922**	38.836	37.962	35.206	36.546	**88.146**	-	19.418	29.567

Table 1: Kendall rank correlation coefficients of the used metrics with SPUR (416 data points), SPUR (+Arjun) (441 data points), UniGen3 (241 data points) and UniGen3 (+Arjun) (309 data points). All of the p-values are lower than 0.001**.**

Answer to RQ1: The number of equivalence classes and the number of variables in the MIS strongly correlate (> 62 for all formulae) with computation time and memory usage of both UniGen3 and SPUR. If the formulae are preprocessed with Arjun, then we find that the highest correlations are with the number of variables, the number of clauses, δ and the size of the MIS.

5.2 RQ2: complexity prediction

We cover here the results regarding formula classification using our trained random forests. We consider binary classification here. We selected the formula processed within the following affordability limits: 30 minutes of computation time and less than 4GB of virtual memory. This selection allowed balanced training data.

Table 2 shows the different Gini importances (i.e. feature importances) of our different metrics in a random forest that contains 1000 instances. The lines where the SAT sampler is suffixed with "(+Arjun)" are the lines where the formulae were first preprocessed with Arjun [42]. The time and memory used for a single Z3 call play a negligible role. The two main features are the number of equivalence classes and the size of the MIS. If however, we use Arjun as preprocessor we observe that the number of variables, the number of clauses, the number of literals and δ seem to be interesting choices as well further confirming our initial correlations. The treewidth has high importance for SPUR (+Arjun) but is expensive to compute, diminishing its value for large formulae.

In Table 3 we explore the F1-scores of a random forest containing 100 instances that were trained on different metrics. The "all" line indicates the predictor trained on all of the metrics. We also use δ' instead of δ in some of the experiments. δ' is defined as $\delta' = \#c - \#v$. While this is only an estimation of δ, our experiments show that it is usually a very good estimation and it is a lot faster to compute as well. As previously, we report sampler results with and without the Arjun preprocessing step. #eqv is always ignored when Arjun is used as a preprocessor. Arjun automatically simplifies the equivalence classes in the formula, thus we find $\#v = \#eqv$ for the preprocessed formulae eliminating the need to computing $\#eqv$. The table entries that involve both Arjun and $\#eqv$ are simply computed by ignoring $\#eqv$. The predictions were done using a leave-one-out strategy and the F1-scores evaluated on the predictions. This means that for every data-point x, we trained a model on the complete data-set excluding x and performed a prediction for x. The predictions are collected in a table and the scores are computed on the prediction table. Table 4 shows the ROC AUCs just like Table 3 shows the F1-scores.

We observe that while the model trained on all features seems to perform best, the model trained on only a fraction of the features perform almost identically. The tables also show that if we were to take only one metric, then the number of equivalence classes is the best unless Arjun is used as a preprocessor in which case δ' and the number of clauses seem to be very good candidates. If we focus on easily computable metrics, then the models that seem most promising are the ones trained on the number of variables, δ' and on the number of equivalence classes. If we preprocess the formulae with Arjun then the number of variables and δ' seem sufficient. Furthermore, we find that using Arjun increases both F1 scores and ROC AUCs.

In Table 5, we reported the F1-scores of decision trees (DT) and random forests (RF) using a different number of instances. The models were trained using #v, δ' and #eqv (if Arjun is not used) and were evaluated using a leave-one-out strategy. We observe that a random forest containing 100 instances performs slightly better than the other models.

Answer to RQ2: We find that the number of equivalence classes alone forms an excellent predictor to classify sampling difficulty according to an affordability budget. Similarly, we observe that if Arjun is used to preprocess a formula, then prediction becomes easier.

5.3 RQ3: URS

Table 6 shows the Kendall rank correlation coefficients for the MiniSAT and Z3 SAT solvers as well as our implementation of BSAT using Z3 and the state-of-the-art model counters D4 and sharpSAT.

All the 488 models have been used for the lines involving Z3, MiniSAT and BSAT as all managed to be sampled in less than 5 hours and with less than 64GB of virtual memory. The BSAT algorithm seems strongly correlated to the size of the MIS as well as the number of equivalence classes but it is even more correlated to the number of variables, clauses, and literals. BSAT does seem close to SPUR and UniGen3 however for the values of the correlation

	#v	#c	#l	tw	δ	#mis	#eqv	time Z3	mem Z3
SPUR	9.811	6.572	4.689	10.947	7.705	13.061	**37.869**	3.367	5.975
UniGen3	7.016	8.327	11.066	2.706	8.774	**27.061**	**31.631**	1.496	1.918
SPUR (+Arjun)	4.023	16.212	**20.819**	**25.342**	17.860	2.962	-	2.951	9.829
UniGen3 (+Arjun)	**27.684**	**20.963**	12.669	4.479	10.531	18.247	-	1.843	3.585

Table 2: Feature importances in a random forest containing 1000 instances

	SPUR	SPUR (+Arjun)	UniGen3	UniGen3 (+Arjun)
#mis	67.469	73.333	91.329	90.995
#eqv	80.981	-	91.254	-
δ'	62.721	85.714	83.082	90.783
#v	60.000	62.400	86.629	93.706
#c	65.497	86.179	85.499	91.469
all	83.950	91.200	97.338	96.759
#v, #c, #l, tw, δ, #mis, #eqv	85.185	92.683	97.904	96.998
#v, #c, tw, δ, #mis, #eqv	83.229	91.057	98.084	96.774
#v, tw, δ, #mis, #eqv	84.472	91.057	97.888	96.998
#v, tw, δ', #mis, #eqv	83.018	92.683	98.467	96.998
#v, δ', #mis, #eqv	83.544	-	97.896	-
#v + δ' + #mis	78.750	89.256	97.142	96.296
#v + δ' + #eqv	83.333	-	94.656	-
#v + δ'	68.674	88.524	89.591	95.833
#v + #c + δ'	66.667	90.164	89.552	95.833
#v + #c + #l + δ'	68.712	88.710	89.888	95.814

Table 3: F1-scores with different features of a random forest containing 100 instances estimated using LOO

	SPUR	SPUR (+Arjun)	UniGen3	UniGen3 (+Arjun)
#mis	80.115	83.886	90.804	83.886
#eqv	87.676	-	90.491	-
δ'	77.587	90.444	81.334	90.444
#v	76.025	78.712	85.264	78.712
#c	79.496	91.829	84.028	91.829
all	89.238	95.283	97.085	95.283
#v, #c, #l, tw, δ, #mis, #eqv	89.957	95.511	97.718	95.511
#v, #c, tw, δ, #mis, #eqv	88.643	94.591	97.970	94.591
#v, tw, δ, #mis, #eqv	89.362	94.591	97.779	94.591
#v, tw, δ', #mis, #eqv	88.171	95.511	98.382	95.511
#v, δ', #mis, #eqv	88.295	-	97.748	-
#v + δ' + #mis	85.891	92.978	96.894	92.978
#v + δ' + #eqv	87.824	-	94.230	-
#v + δ'	80.834	92.863	88.249	92.863
#v + #c + δ'	79.520	93.784	88.279	96.385
#v + #c + #l + δ'	80.487	93.556	88.722	96.328

Table 4: ROC AUCs with different features of a random forest containing 100 instances estimated using LOO

coefficients with the size of the MIS and the number of equivalence classes. We do find that D4 and sharpSAT have very similar correlation coefficients with both SPUR and UniGen3. This would indicate that in practice, the complexity of model counters and uniform random samplers are very close.

For both MiniSAT and Z3 we observe a strong correlation with the number of variables, the number of clauses, and the number of literals. We do not however observe a high correlation with the MIS or the number of equivalence classes. The correlation coefficients seem to be very different between SAT solving and URS. On the other hand, BSAT seems to be a combination of SAT solving and URS in terms of correlations.

	SPUR	SPUR (+Arjun)	UniGen3	UniGen3 (+Arjun)
DT	82.424	89.256	95.635	94.836
RF 10	83.116	89.076	95.000	95.059
RF 100	84.810	88.525	95.419	96.759
RF 1000	83.544	88.333	95.219	96.536

Table 5: F1-scores with different models trained on #v, δ' and #eqv estimated using LOO

	#v	#c	#l	tw	δ	#mis	#eqv	time Z3	mem Z3
time Z3	69.198	72.567	71.901	55.718	71.659	45.977	47.383	100.000	72.171
mem Z3	76.995	81.533	79.994	64.732	80.993	49.527	56.587	72.171	100.000
time MiniSAT	68.033	74.231	74.051	64.682	76.411	45.885	55.390	64.398	73.426
mem MiniSAT	72.642	76.102	74.372	62.203	75.609	46.692	52.144	66.101	80.884
time BSAT	77.798	74.961	75.568	55.571	72.368	67.511	65.485	60.592	66.412
mem BSAT	89.193	86.025	83.541	65.261	82.968	62.486	71.169	66.816	76.081
time D4	54.937	55.866	56.154	45.265	55.966	59.494	70.318	39.511	48.182
mem D4	64.563	63.893	62.376	49.951	62.592	58.345	62.926	47.959	56.842
time sharpSAT	49.315	50.093	51.305	37.973	50.405	60.643	64.033	34.972	41.061
mem sharpSAT	35.865	35.896	36.580	22.315	34.994	49.271	42.844	21.886	25.869

Table 6: Kendall rank correlation coefficients of the used metrics with Z3 and MiniSAT (488 data points), as well as BSAT using Z3 (488 data points), D4 (437 data points) and sharpSAT (416 data points).

Answer to RQ3: SAT solving and URS correlate with different metrics and are thus different tasks. SAT model counting seems to be very close to URS. BSAT seems to be a combination of both SAT solving and URS in terms of correlation.

5.4 Perspectives

Our results demonstrate that the number of equivalence classes in a formula has strong correlations with the computational complexity of sampling. This opens the perspective of increasing sampling efficiency by transforming the input formulae into an equivalent formula with fewer equivalence classes similarly to Arjun.

We also revealed that, though less than the equivalence classes, the MIS also shows strong correlations with sampling complexity. Therefore, efficient ways to compute the MIS and project a formula onto its MIS would also increase URS efficiency. This is demonstrated through the usage of Arjun which further confirms our results. Moreover, Arjun allowed the samplers to solve more instances and increased the performance of our prediction models.

6 THREATS TO VALIDITY

As for any empirical study, there is a number of threats to consider.

Construct Validity. To assess the validity of our findings, we used the Kendall rank correlation coefficient on the existing and our new *#EQV* metric. The Kendall rank correlation coefficient is non-parametric (and therefore agnostic to the data distribution) and was used in the past to establish a relationship between structural metrics and runtime measures [3]. Regarding the evaluation of our random forest predictors, we used both F1-score and Receiver Operating Characteristics (ROC) in order to cope with different classification thresholds. The main reason for this is that we have highly imbalanced data and both metrics react differently to imbalance.

External Validity. We cannot guarantee that our findings generalize to any formula and all tools in each category (sampling, solving, counting). The reason behind this is the lack of general understanding of the complexity of SAT-based tasks [15], which we aim to address with new metrics. To mitigate this threat, we selected a range of SAT formulae from two different sources. They come from SAT Benchmarks used for the evaluation of uniform samplers [7, 8, 13] and feature models representing configurable systems of various types and sizes [1, 37]. In both FM and non-FM categories, formulae encode different types of models: Electonic circuits, algorithmic problems, etc. for the former, and Linux kernels, Unix command line tools or configuration tools [17] for the latter.

7 RELATED WORK

Complexity of SAT problems. As noted by Alyahya *et al.* [3] and Vardi *et al.* [15], studying the complexity of SAT-based tasks is not new. One of the first approaches was to characterise *phase transitions* linked to abrupt changes in solving complexity. Monasson *et al.* offered a structural metric, namely the ratio of clauses to variables [29]. They were able to demonstrate that when this ratio increases, finding solutions for a given randomly generated formula is progressively harder up to a critical value of this ratio past which the formula becomes easy to solve again (often by proving it UNSAT). Alyahya's survey further covers metrics we also used in this study, such as treewidth correlated with solving time [27]. These metrics were not so far assessed for URS techniques. Yet, MIS is expected to play a role in the scalability of sampling [42]. We found that MIS is indeed correlated with solving time and memory consumption but the difficulty of computing MIS is an issue. This motivated us to offer a more scalable metric.

Regarding FM-based formulae specifically, the body of knowledge is more limited. Mendonca *et al.* did not observe such phase transitions for FM formulae: solving was easy throughout the ratio values [28]. Liang *et al.* [26] further confirmed these results on larger industrial FMs. Johansen discusses the implications of these findings for combinatorial interaction testing of software product lines [20]. This study is the first to evaluate and offer metrics for uniform sampling of FM and non-FM formulae, in which we show that direct comparison with solving does not hold.

8 CONCLUSION

To understand the complexity of SAT-based uniform sampling, solving and counting, we have proposed an efficient algorithm to compute the equivalence classes (EQV) of a Boolean formula F. This metric possesses two desirable properties other structural metrics fail to have both: *i)* a strong correlation to the computation time and memory consumption **and** *ii)* its computation scales even on complex formulae, thanks to its ability to exploit parallel computing infrastructures. We showed that EQV can accurately (ROC AUC scores $> 87\%$) predict if a formula F is going to be easy or difficult to sample uniformly.

Furthermore, we showed that preprocessing techniques like Arjun can not only improve the scalability of samplers but also make the performance predictions of said samplers easier and more accurate further motivating the development of efficient preprocessing techniques for URS and model counting.

We also highlighted that EQV helped understand where URS complexity stands compared to two other SAT-based tasks: solving and model counting. We found that, at least in practice, URS is closer to model counting than to SAT solving. On the one hand, this prevents the naive use of standard solvers as uniform samplers. On the other hand, it further motivates research at the intersection of model counting with uniform sampling [42]. We expect our metric as well as Arjun to play a role in this bidirectional relationship, *e.g.,* supporting the development of new knowledge compilation techniques.

ACKNOWLEDGMENTS

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR). Gilles Perrouin is a FNRS Research Associate. Maxime Cordy and Olivier Zeyen are supported by FNR Luxembourg (grants
C19/IS/13566661/BEEHIVE/Cordy and AFR Grant 17047437).

REFERENCES

[1] Mathieu Acher, Gilles Perrouin, and Maxime Cordy. 2021. BURST: a benchmarking platform for uniform random sampling techniques. In *SPLC '21: 25th ACM International Systems and Software Product Line Conference, Leicester, United Kindom, September 6-11, 2021, Volume B*, Mohammad Reza Mousavi and Pierre-Yves Schobbens (Eds.). ACM, 36–40. https://doi.org/10.1145/3461002.3473070
[2] D. Achlioptas, Zayd Hammoudeh, and P. Theodoropoulos. 2018. Fast Sampling of Perfectly Uniform Satisfying Assignments. In *SAT*.
[3] Tasniem Nasser Alyahya, Mohamed El Bachir Menai, and Hassan Mathkour. 2022. On the Structure of the Boolean Satisfiability Problem: A Survey. *ACM Comput. Surv.* 55, 3, Article 46 (mar 2022), 34 pages. https://doi.org/10.1145/3491210
[4] Anonym. 2023. EQV and preprocessing for URS prediction. https://github.com/serval-uni-lu/urs_scal.
[5] Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021.* IEEE, 312–323. https://doi.org/10.1109/ICSE43902.2021.00039
[6] D. Batory, D. Benavides, and A. Ruiz-Cortés. 2006. Automated Analysis of Feature Models: Challenges Ahead. *Commun. ACM* (December 2006).
[7] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In *Tools and Algorithms for the Construction and Analysis of Systems TACAS'15 2015, London, UK, April 11-18, 2015. Proceedings.* 304–319.
[8] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2014. Balancing Scalability and Uniformity in SAT Witness Generator. In *Proceedings of the 51st Annual Design Automation Conference* (San Francisco, CA, USA) *(DAC '14)*. ACM, New York, NY, USA, Article 60, 6 pages. https://doi.org/10.1145/2593069.2593097
[9] Michael Codish, Yoav Fekete, and Amit Metodi. 2013. Backbones for Equality. In *Haifa Verification Conference.*
[10] Maxime Cordy, Mike Papadakis, and Axel Legay. 2020. Statistical Model Checking for Variability-Intensive Systems. In *Fundamental Approaches to Software Engineering - 23rd International Conference, FASE 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12076)*, Heike Wehrheim and Jordi Cabot (Eds.). Springer, 294–314. https://doi.org/10.1007/978-3-030-45234-6_15
[11] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *TACAS'08* (Budapest, Hungary). Springer-Verlag, 337–340.
[12] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. 2015. Money for Nothing: Speeding Up Evolutionary Algorithms Through Better Initialization. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) *(GECCO '15)*. ACM, New York, NY, USA, 815–822. https://doi.org/10.1145/2739480.2754760
[13] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. 2018. Efficient sampling of SAT solutions for testing. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018.* 549–559. https://doi.org/10.1145/3180155.3180248
[14] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *SAT'03*. Springer, 502–518.
[15] Vijay Ganesh and Moshe Y. Vardi. 2021. On The Unreasonable Effectiveness of SAT Solvers. In *Beyond the worst-case analysis of algorithms*, Tim Roughgarden (Ed.). Cambridge University Press, 547—563.
[16] Priyanka Golia, M. Soos, Sourav Chakraborty, and Kuldeep S. Meel. 2021. Designing Samplers is Easy: The Boon of Testers. *2021 Formal Methods in Computer Aided Design (FMCAD)* (2021), 222–230.
[17] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2018. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering* (17 Jul 2018). https://doi.org/10.1007/s10664-018-9635-4
[18] Michael Hamann and Ben Strasser. 2015. Graph Bisection with Pareto Optimization. *Journal of Experimental Algorithmics (JEA)* 23 (2015), 1 – 34. https://api.semanticscholar.org/CorpusID:3395784
[19] Alexander Ivrii, Sharad Malik, Kuldeep S Meel, and Moshe Y Vardi. 2016. On computing minimal independent support and its applications to sampling and counting. *Constraints* 21, 1 (2016), 41–58.
[20] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible. In *Model Driven Engineering Languages and Systems: 14th International Conference, MODELS '11*, Jon Whittle, Tony Clark, and Thomas Kühne (Eds.). Springer, 638–652.
[21] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-based sampling of software configuration spaces. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1084–1094.
[22] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA).* Technical Report CMU/SEI-90-TR-21. SEI.
[23] 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 *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017.* 291–302. https://doi.org/10.1145/3106237.3106252
[24] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. 2018. Propagating configuration decisions with modal implication graphs. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018.* 898–909. https://doi.org/10.1145/3180155.3180159
[25] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler. In *IJCAI*.
[26] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. 2015. SAT-based Analysis of Large Real-world Feature Models is Easy. In *Proceedings of*

the 19th International Conference on Software Product Line (Nashville, Tennessee) *(SPLC '15)*. ACM, New York, NY, USA, 91–100. https://doi.org/10.1145/2791060.2791070

[27] R. Mateescu. 2011. Treewidth in Industrial SAT Benchmarks.

[28] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. 2009. SAT-based analysis of feature models is easy. In *SPLC'09* (San Francisco, California). IEEE, 231–240.

[29] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400, 6740 (1999), 133–137.

[30] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 61–71. https://doi.org/10.1145/3106237.3106273

[31] Jeho Oh, Paul Gazzillo, and Don S. Batory. 2019. *t*-wise coverage by uniform sampling. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*, Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi (Eds.). ACM, 15:1–15:4.

[32] Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne, and Sylvain Peyronnet. 2011. Uniform Monte-Carlo Model Checking. In *Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. 127–140.

[33] Umut Oztok and Adnan Darwiche. 2014. On Compiling CNF into Decision-DNNF. In *CP*.

[34] Daniël Paulusma and Stefan Szeider. 2019. On the parameterized complexity of (k, s)-SAT. *Inf. Process. Lett.* 143 (2019), 34–36.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[36] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product Sampling for Product Lines: The Scalability Challenge. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (2019). https://api.semanticscholar.org/CorpusID:85462202

[37] 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 *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*. 240–251.

[38] Matt Raible. 2015. *The JHipster mini-book*. C4Media.

[39] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*. IEEE Computer Society, Washington, DC, USA, 136–145. https://doi.org/10.1109/RE.2006.23

[40] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. 2018. Knowledge Compilation meets Uniform Sampling. In *Proceedings of International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*.

[41] Mate Soos, Stephan Gocht, and Kuldeep S. Meel. 2020. Tinted, Detached, and Lazy CNF-XOR solving and its Applications to Counting and Sampling. In *Proceedings of International Conference on Computer-Aided Verification (CAV)*.

[42] Mate Soos and Kuldeep S Meel. 2021. Arjun: An Efficient Independent Support Computation Technique and its Applications to Counting and Sampling. *arXiv preprint arXiv:2110.09026* (2021).

[43] Marc Thurley. 2006. sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP. In *Theory and Applications of Satisfiability Testing - SAT 2006*, Armin Biere and Carla P. Gomes (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 424–429.