Exploring the Computational Complexity of Uniform Random Sampling and SAT Counting with Phase Transitions

Olivier Zeyen
Interdisciplinary Centre for Security, Reliability and Trust
Univirsity of Luxembourg
Luxembourg, Luxembourg
olivier.zeyen@uni.lu

Maxime Cordy
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg
Luxembourg, Luxembourg
maxime.cordy@uni.lu

Gilles Perrouin
Faculty of Computer Science
PReCISE, University of Namur
Namur, Belgium
gilles.perrouin@unamur.be

Mathieu Acher
INSA
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
mathieu.acher@irisa.fr

Abstract

Uniform Random Sampling (URS) and Model Counting (#SAT) are two intrinsically linked, theoretical problems with relevant practical applications in software engineering. In particular, in configurable system engineering, URS and #SAT can support studying configurations' properties unbiasedly. Despite the community efforts to provide scalable URS and #SAT tools, solving these problems efficiently remains challenging for a large number of formulae. Contrary to the classical SAT problem, whose complexity has been an object of fundamental studies, little is known about what makes a formula hard to sample from. For the first time, we investigate how phase transitions can explain the practical complexity of sampling. Our results, computed on 11,409 synthetic formulae and 4656 real-world formulae, show that phase transitions occur in both cases, but at a different clause-to-variable ratio than for SAT tasks. We further reveal that low formula modularity is correlated with a higher URS/#SAT time. Overall, our work contributes to a principled understanding of URS and #SAT complexity.

Keywords

Uniform Random Sampling, SAT, Model Counting, Scalability, Controlled Experiments

ACM Reference Format:
Olivier Zeyen, Maxime Cordy, Gilles Perrouin, and Mathieu Acher. 2025. Exploring the Computational Complexity of Uniform Random Sampling and SAT Counting with Phase Transitions. In *29th ACM International Systems and Software Product Line Conference - Volume A (SPLC-A '25), September 01–05, 2025, A Coruña, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3744915.3748473

1 Introduction

Uniform Random Sampling (URS) is the problem of selecting models from a Boolean formula, such that each model gets the same probability of being selected. URS has many applications for software engineering, particularly in analyzing large configurable systems. In software product lines (SPLs), feature models (FMs) define all valid configurations using features and constraints. A product is thus a set of features that satisfy the constraints. Exhaustively testing all configurations is often infeasible, making it useful to sample configurations uniformly to detect bugs at an affordable cost [38, 40, 41]. Moreover, URS has been shown to be effective when searching for an optimal configuration [38].

Model counting (#SAT), which counts the number of models of a Boolean formula, is closely related to URS, sharing similar principles and heuristics. URS can also be reduced to #SAT [39]. Beyond URS, #SAT has various applications in software engineering, including variability reduction [48], variability analysis [12, 46], feature prioritization [48], and bug fix prioritization [28].

URS and #SAT are difficult to solve efficiently, with existing methods struggling to scale to large real-world formulae like the Linux kernel [41]. Unlike traditional SAT, the complexity factors behind URS and #SAT remain underexplored. We argue that understanding these factors is crucial to determining the computational expense of analyzing a formula. Following decades of research in SAT [34, 35], our goal is to identify these factors through systematic analysis, starting with an investigation into whether their complexity can be understood through *phase transitions.*

Phase transitions are abrupt changes in a system's properties due to small variations in a parameter. Phase transitions in SAT, #SAT, and URS are defined as a critical point in a parameter space (e.g., clause-to-variable ratio) where solver behavior changes abruptly, often following an easy-hard-easy pattern—problems are easy to solve before and after the transition, but hard near the critical point [20, 23]. For SAT problems, Mitchell *et al.* [34] observed an easy-hard-easy pattern in the clause-to-variable ratio. As the ratio increases, SAT solver runtime remains low until it spikes near 4.25, then decreases again for higher ratios. This phenomenon, known as the SAT phase transition, has driven research into algorithms focused on instances at this critical point [9, 13, 19, 36, 42], highlighting its theoretical and practical significance [21]. Gupta *et al.* [23] studied phase transitions for knowledge compilation and found that the phase transition for knowledge compilation occurs at a different clause-to-variable ratio than for SAT solving. However, these existing works did not study phase transitions for URS. Moreover, they did not study the effects of community structure on the phase transition, which is known to occur in real-world formulae [4].

ACM ISBN 979-8-4007-2024-6/25/09
https://doi.org/10.1145/3744915.3748473

In this paper, we contribute to a principled understanding of URS and #SAT complexity by studying whether phase transitions also occur in these problems. Our investigations require both experimental analysis (based on controlled experiments and artifacts) and empirical analysis (uncontrolled observations made on existing practices). While empirical observations on real-world formulae are needed to validate conclusions in practice, the low availability and high heterogeneity of these formulae entail an insufficient coverage of all possible structural variations to draw general conclusions. Therefore, we also conduct experiments on synthetic formulae created through systematic and controlled procedures to identify trends and limit cases. Doing so allows us to explore the role of specific characteristics in the complexity of URS and #SAT.

We begin by studying the complexity of model counters and uniform random samplers using k-CNF formulae, where each clause has exactly k literals. k-CNF formulae are commonly used in SAT studies [34], enabling direct comparison with SAT solving and other knowledge compilation studies [23]. We vary the clause-to-variable ratio, a key factor in phase transitions for SAT [34], and set $k = 3$, as in previous SAT studies. We observe phase transitions in both URS and #SAT at a lower ratio (2.00 vs. 4.25). This holds regardless of the formula's community structure [4], though a weaker community structure leads to larger increases in computation time.

Next, we investigate the causes of the phase transitions similarly to [23]. While SAT phase transitions are linked to a sudden change in satisfiability probability, URS and #SAT, which explore all models rather than finding just one, require a different explanation. We find that the complexity of URS and #SAT can be explained by analyzing the number of models relative to the number of variables, known as the solution density [6, 23].

Finally, we empirically analyze real-world formulae to verify if the trends observed in synthetic data hold. Real-world formulae have a heterogeneous structure [51], mixing clauses of varying sizes, which complicates general conclusions. Nevertheless, our observations suggest that phase transitions may not occur in real-world formulae which coincides with the findings in [32].

Altogether, this paper makes the following contributions:

(1) ***The first systematic study of phase transitions in both URS and #SAT.*** Our analysis of 2 samplers, 3 #SAT solvers, and 11,409 synthetic formulae shows that phase transitions occur in these problems, but at a different clause-to-variable ratio than in SAT. Thus, a formula that is easy for a SAT solver may not be easy for URS or #SAT.
(2) ***A novel exploration of the reasons behind the complexity of URS and #SAT.*** Through our in-depth study of formula characteristics and the observation of phase transitions, we bring a fundamental contribution to the understanding of URS and #SAT complexity.

Open science policy. All our experimentation infrastructure is available at the following website: https://anonymous.4open.science/r/urs_phase_transitions-6FBC. The repository contains the artifacts that were used. The repository also includes the resulting CSV files of our experiments.

2 Related Work

As noted by Alyahya *et al.* [3] and Vardi *et al.* [18], studying the complexity of SAT-based tasks is not new. One of the first approaches was to characterize *phase transitions* linked to abrupt changes in solving complexity. Monasson *et al.* offered a structural metric, namely the clause-to-variable ratio [35]. They demonstrated that when this ratio increases, finding models for a given synthetic formula is progressively harder up to a critical value of this ratio. When the ratio exceeds this critical value, the formula becomes easy to solve again (often by proving it UNSAT). Alyahya's survey further covers metrics such as treewidth correlated with solving time [31].

Regarding FM-based formulae specifically, the body of knowledge is more limited. Mendonca *et al.* [33] studied the experimental complexity of SAT-solving of FM-based formulae. The authors studied the clause-to-variable spectrum of formulae similar to feature models. In their studies, the authors failed to observe a phase transition and concluded that FM formulae do not suffer from the SAT phase transition thus explaining the general efficiency of SAT-based analysis of feature models. Liang *et al.* [30] further confirmed these results on larger industrial FMs. The authors found that FMs have a high number of unrestricted variables due to the high variability of FMs. The authors also found that SAT-solvers do little backtracking during search, thus explaining the high efficiency. The authors followed by disabling SAT solver heuristics and found that the solver did not suffer from any performance deterioration while solving FMs. In addition to this extensive analysis, the authors ran a set of simplifications to the formulae and found that they were highly efficient. Some of the instances were solved by the simplification procedure alone. The remaining formulae were small in comparison and thus efficiently solved by state-of-the-art SAT-solvers. Johansen discussed the implications of these findings for combinatorial interaction testing of software product lines [26].

Regarding #SAT and uniform random sampling specifically, the body of knowledge is scarce. Sundermann *et al.* evaluated 21 #SAT solvers on FM-based formulae and computed the correlation between execution time and formula metrics [48]. Plazar *et al.* studied the scalability of two samplers, namely UniGen [11] and Quicksampler [14], but the scalability study is limited to UniGen as Quicksampler is not a uniform sampler [41]. Escamocher *et al.* studied the generation of hard instances for #SAT [16]. Huang *et al.* [25] observed a phase transition for knowledge compilation, similarly to [5, 7]. Gupta *et al.* studied the phase transition in more depth for knowledge compilation to d-DNNF, SDD, and OBDD [23]. Similarly, Gao *et al.* [20] studied phase transitions for knowledge compilation as well. However, Gao *et al.* do not study the phase transitions for URS, the community structure, and its influence on the phase transition or the solution density proposed by Gupta *et al.* [23]. Overall, none of these studies explore phase transitions for uniform random sampling, and none of the studies explore the influence of community structure on solving times.

To the best of our knowledge, our study is the first to explore phase transitions for both uniform random sampling and #SAT. In addition to phase transitions, we study the effects of community structure and the links with real-world formulae.

3 Background

3.1 Boolean Formulae

A Boolean formula F is defined over a set of Boolean variables $Var(F)$, which can evaluate to either 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.

An assignment is a set of literals such that $\forall l \in a : (\neg l \notin a)$. An assignment a is complete if $\forall v \in Var(F) : (v \in a \vee \neg v \in a)$. A model m of F, noted $m \models F$, is a complete assignment of $Var(F)$ such that F evaluates to true. We define R_F as the complete set of models of F such that $m \models F$ if and only if $m \in R_F$. Thus, $|R_F|$ is the number of models for F. An assignment a is partial if $Var(a) \subset Var(F)$. We denote by $|a|$ the number of assigned variables.

A formula F is in negational normal form (NNF) if the negation only appears directly in front of variables. A clause is a disjunction of literals. A formula is in conjunctive normal form (CNF) if it is written as a conjunction of clauses. In other words, a formula is in CNF if it is written as $F = \bigwedge_{c_i} \bigvee_{l_i \in c_i} l_i$ with $\bigvee_{l_i \in c_i} l_i$ a clause and l_i a literal. A formula is a k-CNF if every clause has exactly k distinct literals. A formula F in conjunctive normal form can be represented as a set of clauses, and a clause can be represented as a set of literals. For a CNF formula F, we denote by $|F|$ its number of clauses and by $|Var(F)|$ the number of variables that occur in it.

We next define three common problems over Boolean formulae, i.e. SAT solving, model counting, and URS. **SAT solving** is the problem of determining whether R_F is non-empty. **Model counting (#SAT)** is the problem of computing $|R_F|$. Finally, **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.

3.2 Community Structure of Boolean Formulae

An undirected weighted graph G is defined as a pair $G = (V, w)$, where V is the set of nodes and w is the edge-weight function defined as $w : V \times V \longrightarrow \mathbb{R}^+$. Because G is undirected, we have $w(x, y) = w(y, x)$.

The Variable Incidence Graph (VIG) [22] of a CNF formula F is the undirected weighted graph whose nodes are the variables of F. There exists an edge between two variables if they both appear in a clause c. To give the same relevance to all clauses, we define the weight of an edge between nodes x and y as $w(x, y) = \sum_{c \in F, x \in c \wedge y \in c} \frac{1}{\binom{|c|}{2}}$, with $|c|$ the number of literals in the clause.

A formula has a community structure if we can split the variables into at least two groups such that we have a higher number of clauses that connect variables within a group than clauses that connect multiple groups together. This is an interesting property for model counting and sampling because if we have a formula that does not have any connection between the groups (i.e. no clauses connecting groups), then we can compute the model count (or sample) of each group separately and compute the product to obtain the final result. Thus, if we have a high community structure, we expect that the algorithm will finish faster.

To measure the community structure of a formula F we will use the notion of modularity Q as defined in [4, 22, 37] computed on the VIG G of the formula F.

The modularity of a graph G is defined for a given partition C as follows:

$$Q(G, C) = \sum_{C_i \in C} \frac{\sum_{x,y \in C_i} w(x, y)}{\sum_{x,y \in V} w(x, y)} - \left(\frac{\sum_{x \in C_i} deg(x)}{\sum_{x \in V} deg(x)} \right)^2$$

The modularity of a graph is $Q(G) = max\{Q(G, C)|C\}$ for any partition C. Computing the modularity of a graph is NP-hard [8], thus most methods usually approximate a lower-bound of Q. The modularity of a graph will be in the range $[0, 1]$ [4] with one meaning a very strong community structure, and zero meaning that the graph is fully connected.

4 Objectives and Methodology

We detail below our research methods, the protocol to prepare the dataset (Boolean formulae), the samplers and model counters we used, and our computing infrastructure.

4.1 Research Questions and Methods

To achieve a principled understanding of URS and #SAT problem complexity, we explore the phenomenon of phase transitions [34] similarly to SAT problems and knowledge compilation [23]. Mitchell *et al.* showed in [34] that the solving difficulty of random k-CNF formulae changes drastically across the clause-to-variable spectrum. Indeed, they found that formulae are much harder to solve for a part of this spectrum. Around a clause-to-variable ratio of 4.25 (for $k = 3$), the formulae are exponentially harder to solve. This discovery has a major importance as it shows that some of the synthetic formulae drawn from the same distribution are much harder to solve than others. This motivated the development of algorithms that perform better near the phase transition. Similarly, Gupta *et al.* [23] investigated phase transitions in knowledge compilation and identified a phase transition, though at a different clause-to-variable ratio compared to SAT solving. Even though phase transitions have been studied for #SAT and knowledge compilation [7, 20, 23, 25], they remain a mainly observed phenomenon; as such, no formal definition is available. Because available real-world formulae are too scarce and heterogeneous to draw general conclusions (regarding structural properties such as the number of variables/clauses), our study combines experimental analysis (using synthetic artifacts under experimental control) with empirical analysis (based on real-world artifacts collected in the literature). These are necessary to draw theoretical (general) conclusions and validate them in practice.

Our study globally aims to answer three research questions:

- **RQ1:** Do phase transitions generally occur in synthetic URS and #SAT problems?
- **RQ2:** What are the reasons for the phase transitions in synthetic formulae?
- **RQ3:** Are phase transitions also observed on real-world formulae?

To draw general conclusions for RQ1 and RQ2, we rely on controlled experiments. To complement our findings with empirical observations, we answer RQ3 using real-world data.

4.2 Data Preparation

4.2.1 Synthetic Formulae. For our experimental analysis, we conduct controlled experiments with 11,409 formulae that we synthesize by varying multiple structural characteristics (number of variables, number of clauses, clause size, modularity, etc.). Thus, we can observe finely complex phenomena like phase transitions.

To generate our datasets, we use the classical k-CNF generation [34] and the community attachment model from [22]. The classical model enables us to control various parameters: number of variables v, number of clauses n, and clause size k. It then generates a clause by selecting k unique variables and negating them with a $\frac{1}{2}$ probability and add the clause to the set of clauses. We use this model to generate a dataset containing 2,162 synthetic formulae, which will be used to answer RQ2.

The community attachment model from [22] gives additional control over the desired modularity Q of the formula. Controlling Q enables us to experiment on how it affects URS and #SAT time. The community attachment model requires specifying an initial number of communities c (groups of variables that are mostly dependent on each other but may have some dependencies with other communities). The generation then starts by splitting the variables into (roughly) equally-sized communities. It iteratively chooses to generate a clause with variables of a given community with probability $P = Q + \frac{1}{c}$; otherwise, all of the variables of the clause to be generated will be part of distinct communities. We use this model to generate a dataset containing 9,247 formulae, which will be used to answer RQ1.

4.2.2 Real-World Formulae. We collected multiple datasets from Lagniez *et al.* [29], Soos [44], Sundermann *et al.* [47], and Plazar *et al.* [41] to study whether our observations made on synthetic formulae also transfer to the real world.

The Lagniez *et al.* [29] dataset is a diverse dataset containing 1979 formulae. The dataset contains diverse problems ranging from Bayesian networks to digital circuits and configuration. This dataset also contains handmade and random formulae. The Soos [44] dataset contains 1896 formulae from various sources, including the Model Counting Competitions. The Sundermann *et al.* [47] dataset consists of 278 formulae, most of which come from the configurable software domain. The dataset contains multiple versions and variants of each formula. To avoid having too many similar formulae, we restricted our experiments to the most recent version and variant of each formula. The Plazar *et al.* [41] is a dataset of 503 formulae consisting of a feature model benchmark (133 formulae) as well as other formulae collected from [14].

Overall, the dataset includes formulae featuring between five and 1,370,369 variables, and between ten and 5,798,978 clauses. The clause-to-variable ratio of the formulae ranges from 0.46 to 290.93 with an average value of 3.41 and a median value of 2.5.

4.3 URS and #SAT Tools

We selected diverse counters and samplers to explore phase transitions for different URS and #SAT algorithms.

UniGen3 [45]: a hashing-based algorithm that generates samples in a nearly uniform manner with strong theoretical guarantees: it either produces samples satisfying a nearly uniform distribution or produces no sample at all. These guarantees come at a cost: the hashing-based approach requires adding large clauses to formulae so they can be sampled.

SPUR [2]: SPUR is a uniform sampler built on top of sharpSAT [49]. SPUR exploits how sharpSAT walks through all the models of a formula to produce uniform samples. SPUR is one of the few samplers guaranteeing uniformity theoretically.

sharpSAT [49]: a state-of-the-art model counter based on an exhaustive DPLL algorithm with advanced component caching.

D4 [29]: a state-of-the-art compiler that compiles a CNF formula into a decision-DNNF, which is an NNF. The new NNF allows for fast model counting and model enumeration.

McTW [17]: is a model counter based on dynamic programming on treewidth decompositions of a primal graph constructed for a given CNF formula [43].

We chose SPUR [2] and UniGen3 [45] as samplers as they both offer theoretical guarantees regarding their uniformity. Other sampling tools exist, *e.g.,* [14]. These alternatives offer no uniformity guarantee or rely on other tools such as sharpSAT [49] or D4 [29]. We chose sharpSAT [49] as a model counter because it is the fastest model counter to date [48]. We also chose D4 [29] as it is a model counter based on knowledge compilation and decision-DNNF which is slightly different from sharpSAT. Moreover, D4 [29] will serve as a sanity check for our experimental setup because it has already been studied by Gupta *et al.* [23] We would like to highlight that a d-DNNF can be extracted from the execution trace of a model counter based on exhaustive DPLL. Therefore, we do not expect the results between D4 and sharpSAT to deviate significantly. Similarly, the results for SPUR and sharpSAT should be very close given their similarities. Finally, to have better diversity, we tested McTW [17] because it is an algebraic-based model counter [48].

4.4 Infrastructure

We used a computing facility containing 354 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 set a timeout of five hours and a limit of 64GB of virtual memory for counting and sampling experiments. Additionally, we requested 1000 models for each sampler.

5 RQ1: Phase Transitions

To observe phase transition occurrences in URS and #SAT problems, we analyse the execution time of different sampling and counting tools on formulae systematically synthesized. Inspired by the work of Mitchell *et al.* on phase transitions in SAT problems [34], we aim to observe phase transitions on the clause-to-variable ratio (i.e. $|F|/|Var(F)|$) spectrum. Due to space restrictions, we only show the results for SPUR and UniGen3. All of our results are available on our companion GitHub [50].

5.1 Setup

We generate 3-CNF formulae defined over a set of 75 variables, with community structure. We set $k = 3$ because this clause size is typically used to analyze SAT problems as it is the smallest non-trivial value for k. Indeed, $k = 1$ and $k = 2$ are not enough to create non-trivial dependencies between variables, whereas setting a value higher than 3 drastically increases sampling/solving time – as we also illustrate in RQ2. We set the number of variables to 75 because we experimentally determined this was a good trade-off for sampling/counting time (more variables reduce the impact of runtime noise in our results, but more than 75 was computationally prohibitive on our infrastructure).

Because we use 3-CNF formulae, we can compare our findings on URS and #SAT with the clause-to-variable ratio at the phase transition peak of SAT solving in [34]. In the SAT experiments of Mitchell *et al.* [34], solvers' execution time peaks at $|F|/|Var(F)| = 4.25$ and it corresponds to the ratio at which half of the 3-CNF formulae with this ratio are unsatisfiable. For lower ratio values (respectively higher) most of the formulae were satisfiable (respectively unsatisfiable). Since URS and #SAT on unsatisfiable formulae (or, more generally, on formulae with few models) can be efficiently reduced to one (or few) call(s) to a SAT solver, we limit the clause-to-variable ratio to 10 (i.e. about twice the ratio at which SAT complexity is maximal). We also removed accidentally generated unsatisfiable formulae (i.e., with $|R_F| = 0$). Thus, we set the number of clauses from 1 to 750 by a step of 1 (to get a clause-to-variable ratio from 1/75 to 10) and we generate five formulae per number of clause value (to account for random factors in the way clauses are generated).

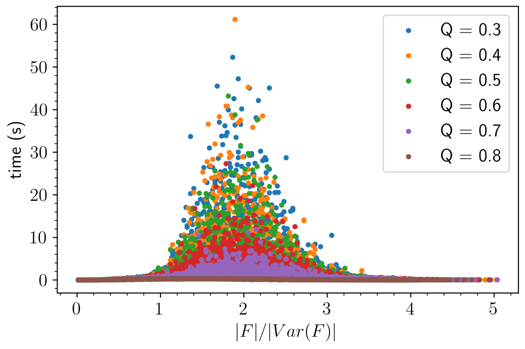
Finally, using the community attachment model from [22], we repeat the generation with different target modularity values Q (ranging from 0.3 to 0.8 with a 0.1 increment as these values have been shown to be common in [4] and in our experiments) to observe the impact of variable dependencies. This yields a total of 9,247 formulae. The community attachment model requires setting the desired number of communities. We choose to use 5 communities, as 15 variables per community should keep the probability of having an unsatisfiable community sufficiently low (considering that a community requires at least 2^k clauses on k variables to be unsatisfiable because a k-CNF formula expressed on k variables requires at least 2^k clauses to be unsatisfiable). For the sake of generality, our companion GitHub [50] page includes result plots for other numbers of communities.

5.2 Results

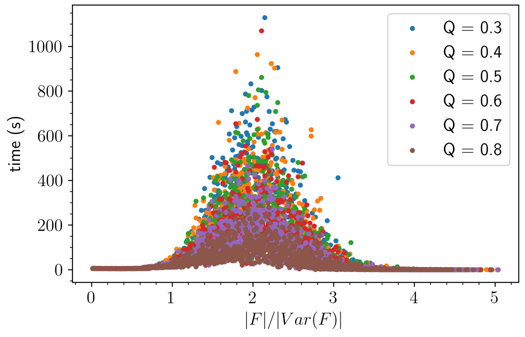
Figure 1 shows the execution time for SPUR and UniGen3. The graphs for each URS and #SAT tool for all generated formulae are available on our companion GitHub [50]. The x-axis is the clause-to-variable ratio, while the y-axis is the execution time. Each data point represents a generated formula, and each color corresponds to a different modularity value targeted by the generation algorithm. We only plot the values that have a clause-to-variable ratio in the range $[0; 5]$ because the formulae outside this range were unsatisfiable and therefore removed from the dataset.

We observe that phase transitions indeed occur for all tools (these are the "peaks" we observe in each plot). Contrary to the SAT problem [34], the hard instances are at a wildly different clause-to-variable ratio, i.e., around 2. URS and #SAT share the same clause-to-variable ratio for the hard instances, with the exception of the McTW model counter, whose performance is generally worse than other #SAT tools. The phase transitions that we observe align with the phase transitions observed in [20, 23, 25]. However, the phase transitions observed in [5, 7] are shifted toward smaller clause-to-variable ratios, which may be due to better optimizations of the algorithms. A more surprising result is that the phase transition observed for UniGen3 is located at a similar clause-to-variable ratio than the phase transitions observed in our study and in [23] even though UniGen3 has a different algorithm and a different purpose.

We also observe that lower target modularity coincides with a taller peak in execution time during the phase transition, indicating that tools exploiting modularity may better resist the complexity of URS and #SAT during the phase transition. A particular observation of UniGen3 – which fundamentally exploits modularity less than other tools because the constraints added by the algorithm to the formula break the community structure – corroborates this finding: its execution time peaks higher for high modularity values than the other tools. Similar results have been shown for SAT solving [22] where SAT solvers specialized in industrial formulae performed better on instances with a high modularity.



(a) SPUR



(b) UniGen3

Figure 1: RQ1: Phase transitions occur in URS for 3-CNF formulae. A higher formula modularity decreases the height of the peak.

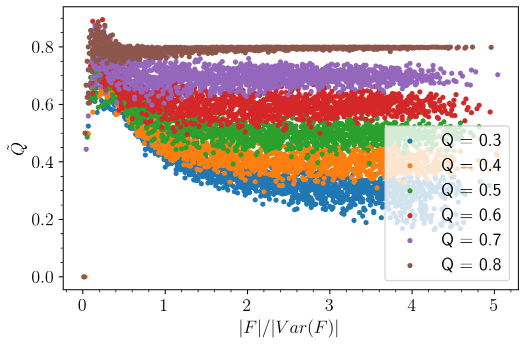
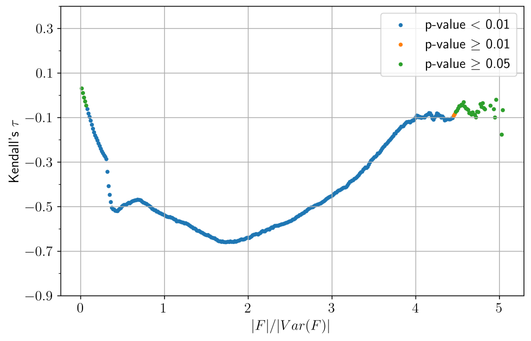
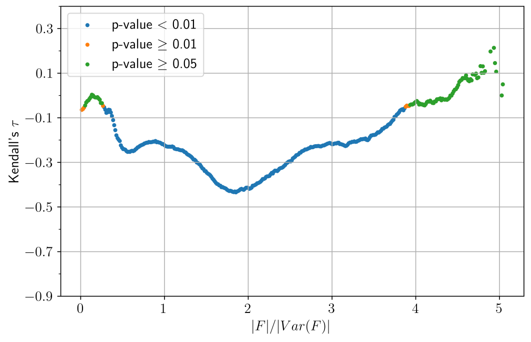


Figure 2: RQ1: Modularity of the 3-CNF formulae (y-axis) w.r.t. their clause-to-variable ratio (x-axis).

To complete these observations, we show in Figure 2 lower bounds to the actual modularity of the generated formulae across the clause-to-variable ratio spectrum. The colors indicate the target



(a) SPUR



(b) UniGen3

Figure 3: RQ1: Kendall's τ coefficients computed between $\tilde{Q}$ and the execution time in a sliding window accros the clause-to-variable ratio spectrum.

modularity parameter given to the community attachment model in [22], which may differ from the actual modularity given the heuristic-based nature of the model. The actual modularity values were computed by using the label propagation algorithm [4] and is denoted by $\tilde{Q}$. This algorithm computes a lower bound for the actual modularity and, being based on label propagation, has stochastic components. Thus, to mitigate stochastic effects and obtain a bound close to the real modularity value, we repeat the modularity computation 100,000 times and take the highest value returned across all those runs.

Figure 2 shows that formulae with a low clause-to-variable ratio tend to have a high actual modularity (in particular, higher than the desired modularity specified in the generation algorithm) as is also shown in [22]. This is also one of the reasons why the phase transition does not occur at these lower clause-to-variable ratios.

Finally, to further confirm our results, we compute the Kendall rank correlation [27] coefficients between the computed modularity $\tilde{Q}$ and the execution time of the different tools. We use the Kendall rank correlation [27] because the relationship is unknown. Therefore, a rank correlation is better suited. Because of the observed phase transition, we find that computing Kendall's τ gives poor results. We thus decide to compute Kendall's τ over a sliding window. In other words, we compute Kendall's τ at a clause-to-variable ratio of x with the formulae that have a clause-to-variable ratio in the range $x \pm \varepsilon$. In our case, we set $\varepsilon = 0.3$. The results are shown in Figure 3, where we plot the Kendall rank correlation coefficients across the different clause-to-variable ratios. The values shown in orange have a p-value higher than or equal to 0.01 and the values shown in green have a p-value higher than or equal to 0.05. We observe strong negative correlations across most of the clause-to-variable ratio spectrum with low p-values. This means that an increase in modularity is correlated with a decrease in computation time, as previously observed. Moreover, almost all of the samplers and model counters have strong correlations. The only sampler with lower correlation coefficients is UniGen3, which still shows a moderate correlation of -0.40, close to the observed phase transition. sharpSAT also exhibits a more surprising behaviour with positive correlations after a clause-to-variable ratio of 4. However, these correlations have high p-values. We thus conclude that community structure has a positive impact on the computation time of uniform random samplers and model counters.

RQ1 – Conclusions: Our results confirm the existence of phase transitions for URS and #SAT on 3-CNF formulae, which occur at different clause-to-variable ratios (starts at ≈ 1 and peaks at ≈ 2) than for classical SAT (peaks at 4.25). A higher formula modularity decreases the height of the peak. Formulae with a low clause-to-variable ratio (< 1) have a higher modularity, which might explain why URS and #SAT are easier for those formulae.

6 RQ2: Reasons for Phase Transitions

We investigate the reasons behind the occurrences of phase transitions in URS and #SAT problems. In the case of SAT, the phase transition peaks at a 4.25 clause-to-variable ratio [34], which corresponds to the ratio at which half of the 3-CNF formulae with this ratio are unsatisfiable. SAT solvers typically work by exploring a search tree where branches represent variable assignments [15]. Formulae with a low number of clauses per variable necessitate few assignments to be solved (it is easy to find a model), whereas formulae with a large number of clauses quickly lead to unsatisfiable branches (it is easy to conclude on the absence of models). At the ratio of 4.25, the SAT solver has to explore many deep branches to conclude the (un)satisfiability of the formula [34].

By contrast, #SAT (and by extension URS, which relies on the same algorithmic principles), requires exploring all branches of the search tree to complete the counting, an exploration known as "exhaustive DPLL". Fortunately, the model counter can prune a part of a branch that has become trivial to solve. This can happen, e.g., if all of the constraints are satisfied by the current assignment a, in which case the current model count is $2^{|Var(F)|-|a|}$. Thus, the complexity of URS and #SAT is intrinsically linked with the minimal number of variables necessary to have at least as many models as the initial formula. In the above example, the minimal number of variables necessary to express the models in the branch is $(|Var(F)| - |a|)/|Var(F)|$.

Gupta *et al.* [23] generalize this idea and hypothesize that *what matters in knowledge compilation complexity is the solution density*. In other words, the ratio of the minimal number of variables necessary to have at least as many models divided by the number of variables of the initial formula. We extend the hypothesis to URS and explore the ratio $r = log_2(|R_F|)/|Var(F)|$. The logarithm of

Table 1: RQ2: Maximum execution time (in seconds).

Tool	$k = 3$	$k = 4$
D4	8.6	3239.97
sharpSAT	0.42	230.49
McTW	207.5	18000.0
SPUR	1.13	542.22
UniGen3	12.07	500.23

$|R_F|$ expresses the minimal number of variables needed to encode $|R_F|$ models. Dividing this by the total number of variables allows us to have a normalized ratio.

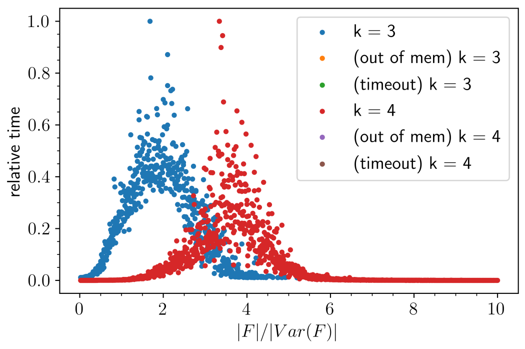
We note that r is a semantic metric as it necessitates computing $|R_F|$, which may not always be feasible. However, our objective here is not to provide a metric to predict occurrences of phase transitions (the clause-to-variable ratio, which is always easy to compute, would be a more appropriate metric for that). Instead, we aim to study the phenomena behind phase transitions and our findings (which corroborate Gupta et al.'s earlier investigations [23]) indicate that r is a more effective metric for this purpose. Note that, since we exclude formulae with no model in our experiments, r takes a value between 0 and 1; moreover, we have $log_2(|R_F|) \leq |Var(F)|$ since $|R_F| \leq 2^{|Var(F)|}$.

6.1 Setup

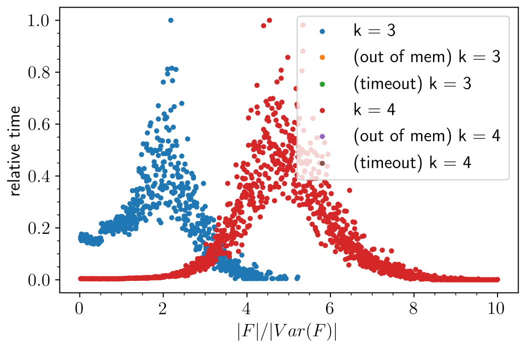
If our hypothesis is true, repeating our RQ1 experiments while setting $k = 4$ (instead of 3) should yield a shift in the observed phase transition. This is because for the same ratio $|F|/|Var(F)|$, increasing the number of literals in each clause yields a higher model count, as a larger clause is more likely to be satisfied by a random assignment. Thus, a higher clause-to-variable ratio should be necessary to observe the phase transition for $k = 4$ than for $k = 3$. To verify this, in addition to 3-CNF formulae, we also generate 4-CNF formulae. Because raising k from 3 to 4 significantly increases the URS and #SAT's tools execution times, we limit formulae to 50 variables. We obtain 663 satisfiable formulae for $k = 3$ and 1499 formulae for $k = 4$ by using the k-CNF generation algorithm proposed by [34]. We have more formulae for $k = 4$ because a higher clause-to-variable ratio is required to observe unsatisfiable formulae. To illustrate this increasing complexity, Table 1 reports the maximum execution time (in s.) of all tools across all formulae with $k = 3$ and $k = 4$; we observe that McTW has reached the timeout of five hours.

In Figure 4, we study the execution time of SPUR and UniGen3 (the remaining graphs are available on our companion GitHub [50]) on all formulae with respect to their clause-to-variable ratio (x-axis). On the y-axis, we normalize execution time based on the maximum time taken by each tool for $k = 3$ and $k = 4$ separately, based on the values reported in Table 1. This normalization helps us compare the trends for $k = 3, 4$, which would not be feasible otherwise due to large differences in execution time when increasing k.

In this figure, we observe that for $k = 4$ the phase transition indeed peaks at larger ratios for SPUR (3.5), sharpSAT (3.5), and UniGen3 (5), whereas for $k = 3$, the execution time peaks at a ratio of 2. There is also a large difference for McTW (3 vs 4.5). For D4, the two clause-to-variable ratios are closer to each other, though



(a) SPUR



(b) UniGen3

Figure 4: RQ2: Phase transitions w.r.t. the clause-to-variable ratio, on 3-CNF and 4-CNF formulae.

$k = 4$ observably yields a peak at a higher ratio. This observed shift is the first indication that our hypothesis indeed holds.

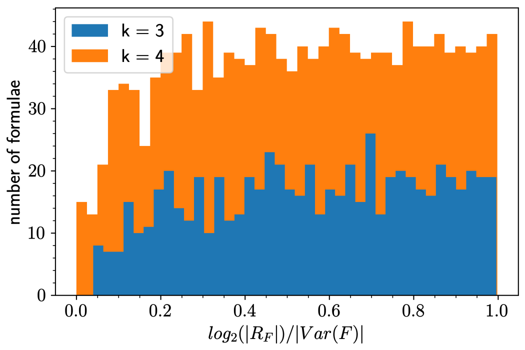
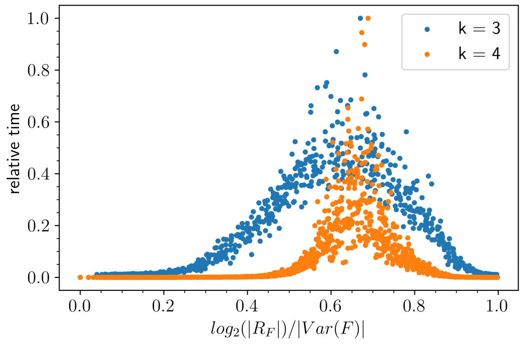
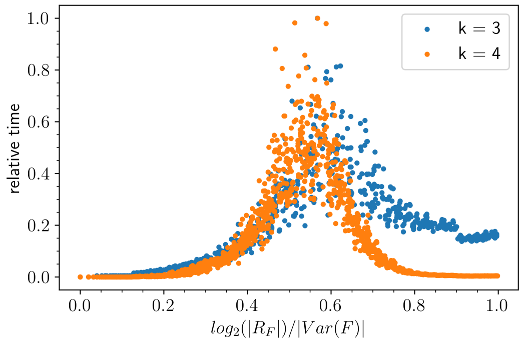


Figure 5: RQ2: Distribution of $log_2(|R_F|)/|Var(F)|$ with respect to k.

To confirm our hypothesis, we study the relationship between URS and #SAT execution time and the new ratio $r = \frac{log_2(|R_F|)}{|Var(F)|}$. We compute $|R_F|$ with D4 [29] for every formula. Because our formula synthesis algorithm is not driven by r or $|R_f|$, we first check that our population of formulae is sufficient to conduct analyses based on r (i.e., the formulae are sufficiently uniform with respect r). Figure 5 shows the distribution of r for the generated 3-CNF and 4-CNF. We observe that the distribution is balanced, except for a small deficit when approaching $r = 0$. This can be explained by the fact that at such r values, a synthesized formula has a higher probability of being unsatisfiable. Fortunately, this bias in the population does not affect the phase transition, which should occur far from $r = 0$ (as later confirmed in our results).



(a) SPUR



(b) UniGen3

Figure 6: RQ2: Phase transitions w.r.t. $r = log_2(|R_F|)/|Var(F)|$ on 3-CNF and 4-CNF formulae.

6.2 Results

Figure 6 plots the tools' normalized execution time (y-axis) (all the graphs are available on our companion GitHub [50]) with respect to r (x-axis). We observe that the hard instances for SPUR and sharpSAT occur at a ratio $r = log_2(|R_F|)/|Var(F)|$ close to 0.65 for both 3-CNF and 4-CNF formulae, whereas the value r for UniGen3 is between 0.55 and 0.6. As for D4 and McTW, we also observe a phase transition, though the peaks deviate slightly between $k = 3$ and $k = 4$ (from 0.68 to 0.74 for D4 similarly to [23], from 0.40 to 0.55 for McTW).

To evaluate whether r provides a better characterization of the phase transition than the clause-to-variable ratio, we compute the **mean absolute error (MAE)** in each case. As a reference, we define a function $f(x)$, computed on the dataset corresponding to $k = 4$, where each value x is associated with the mean of all data points within the interval $x \pm \varepsilon$. Specifically, we use $\varepsilon = 0.3$ when analyzing the clause-to-variable ratio, and $\varepsilon = 0.028$ when analyzing r. These values of ε were chosen to reduce noise-induced fluctuations while preserving the overall trend of the data. In both cases, the size of the sliding window was adjusted to include, on average, the same number of data points—approximately 86 ± 1.

The function $f(x)$ is then normalized to lie within the interval $[0, 1]$. Finally, we compute the MAE between the normalized reference function $f(x)$ and the corresponding dataset to compare the performance of r and the clause-to-variable ratio in describing the phase transition.

Table 2: RQ2: Mean absolute errors of the datasets with the $k = 4$ dataset as reference.

	clause-to-variable ratio		r	
Tool	$k = 3$	$k = 4$	$k = 3$	$k = 4$
D4	0.368	0.109	0.175	0.109
sharpSAT	0.353	0.113	0.158	0.110
McTW	0.185	0.054	0.299	0.057
SPUR	0.377	0.116	0.138	0.117
UniGen3	0.314	0.119	0.184	0.119

Table 2 reports the computed MAEs for both the clause-to-variable ratio and the parameter r, for $k = 3$ and $k = 4$. The left half of the table corresponds to the results shown in Figure 4, while the right half relates to Figure 6.

The MAE values for $k = 4$ serve as a sanity check, as the reference function $f(x)$ is constructed from the $k = 4$ dataset. We observe that for $k = 3$, the MAE consistently decreases when switching from the clause-to-variable ratio to r, suggesting that the apparent shift between the curves for $k = 3$ and $k = 4$ is reduced when using r. This supports the conclusion that r more accurately aligns the phase transitions across different values of k.

The only exception is observed in the case of McTW, where the MAE increases when switching from the clause-to-variable ratio to r. We attribute this deviation to the large number of timeouts encountered by McTW, which may affect the accuracy of the underlying data.

Overall, the results indicate that r provides a better description of the phase transition than the clause-to-variable ratio.

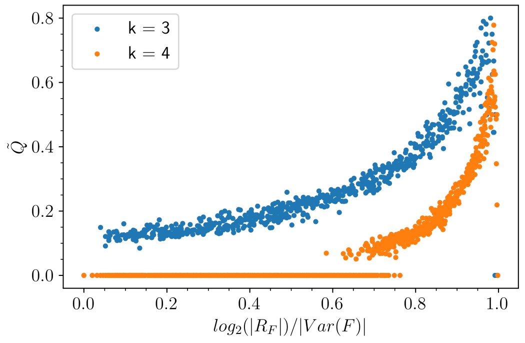


Figure 7: RQ2: Modularity w.r.t. $r = log_2(|R_F|)/|Var(F)|$.

Discussion. Our observations seem to indicate that the hard instances of URS and #SAT need model spaces R_F that are large but with sufficient differences between each model (in terms of assignments). In other words, instances with a high r (equivalently, a low clause-to-variable ratio) are easy to solve because they are underconstrained and quickly generate unconstrained (free) variables during the DPLL exploration, making parts of some search branches trivial to count. By contrast, instances with a low r (or high clause-to-variable ratio) are overconstrained, and their model count is low, which is, therefore, easy to enumerate and count. The instances in between do not allow for an effective processing because the models are too different from each other while the large $|R_F|$ prohibits model enumeration.

A complementary explanation relates r to modularity. Figure 7 shows the actual modularity of the 3-CNF and 4-CNF formulae across the r spectrum. As before, we compute modularity with the label propagation algorithm in [4] repeated 100,000 times and took the maximum modularity value (maximum lower bound). We observe that a high ratio r coincides with a high modularity. This indicates that we have a strong community structure in the formulae and that the solvers would need to satisfy fewer clauses before having entirely disjoint communities. These disjoint communities can then be processed independently, which is D4s strategy [29]. Lower values of r have a low modularity as these formulae are overconstrained.

RQ2 – Conclusions: Phase transitions in URS and #SAT are better described by the ratio of variables required to encode the space of models. Formulae with a high ratio are easy to solve because they are underconstrained, enabling URS and #SAT to take shortcuts and consider unconstrained variables separately. Conversely, those with a low ratio r have few models that are fast to enumerate. The phase transition is due to intermediary situations wherein the number and diversity of models impede both model counting and effective solving.

7 RQ3: Real-World Formulae

7.1 Setup

Our last RQ analyses real-world formulae. Unlike synthetic k-CNF formulae, real-world formulae are likely to include constants and variables that do not appear in clauses (this typically happens, e.g., to formulae derived from feature models because some features may be mandatory and other features may not have any constraints) [30]. To avoid noise in our computed ratios, we first pre-process the formulae to eliminate these constants and unconstrained variables (without altering the formulae semantics). This modification does not affect our analysis of the tools' execution time since the underlying algorithms also remove them during their pre-processing. Furthermore, we also remove redundant clauses, i.e., clauses that are subsumed by another. Assume we have $c_i, c_j \in F$ with $i \neq j$ and $c_i \implies c_j$. Then we only need to consider c_i and can safely ignore c_j because c_i ensures that c_j is satisfied. We thus compute the size of the formula $|F|$ as the number of clauses minus the number of redundant clauses. $|Var(F)|$ is computed as the total number of variables minus the number of variables that do not appear in any (non-redundant) clause. Both the URS and #SAT tools were executed on the original formulae.

Our statistics on the capacity of the URS and #SAT tools to analyze real-world formulae indicate that URS and #SAT tools have approximately the same success rate with respect to timeouts and out-of-memory exceptions. UniGen3 performed significantly worse, with only 2510 formulae that were successfully processed. The other tools processed with success between 3034 and 3996 formulae, with McTW processing successfully 3034 formulae and D4 processing successfully 3996 formulae.

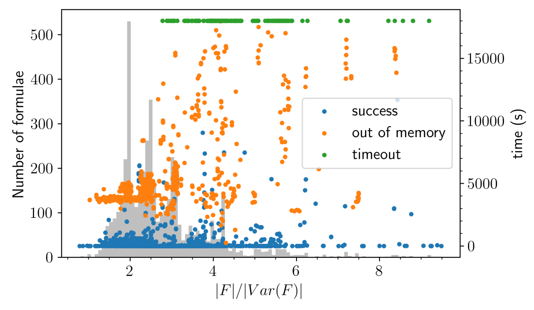
7.2 Results

Figure 8 shows the execution times of the different model counters and samplers on our real-world formula dataset. The x-axis denotes the clause-to-variable ratio. The reason we use this ratio and not r is because model counting (which is required to compute r) is intractable for a significant number of formulae in our dataset; discarding these formulae would leave us with a smaller number of formulae to conduct our analysis. Additional Figures using r are available on our companion GitHub [50].

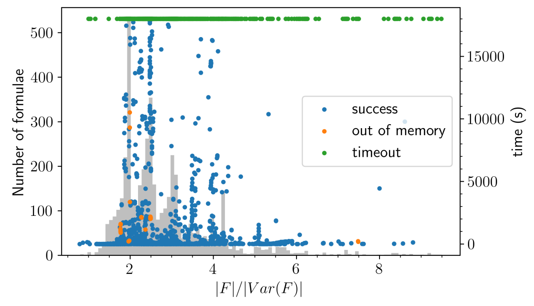
The double y-axis reports two measures: the number of real-world formulae (left-hand side scale) and the execution time (right-hand side scale). Thus, the grey histogram shows the frequency of each clause-to-variable ratio among the set of formulae, while the colored dots represent the execution time of each formula on the considered tool. The colors distinguish the formulae that were processed with success (blue), the formulae for which the model counter or sampler ran out of memory (orange), and the formulae for which the model counter or sampler reached the 5-hour timeout (green).

In Figure 8, we see that while there are some clusters, overall, patterns for phase transitions are difficult to observe. To complement this view, we show in Figure 9 the cumulative distribution of the total number of formulae, the number of formulae processed with success and the number of formulae that could not be processed with respect to their clause-to-variable ratio. While there do seem to be spikes in the line representing failure (the green line), these seem to align with the total number of formulae as shown in the histogram in Figure 8. Generally, the absence of clearly observable phase transitions, coupled with the significant numbers of formulae intractable for URS, indicates that there are additional factors in feature-model formulae that explain their hardness. This means that specific applications of URS to feature models necessitate dedicated complexity studies, as results valid for synthetic formulae are not enough to explain this complexity. Furthermore, the lack of a phase transition for URS and #SAT in real-world formulae is consistent with the findings of Mendonca *et al.* [32], who did not observe a phase transition for SAT in feature models.

RQ3 – Conclusions: In the case of feature models, the observation of phase transitions is blurred by additional complexity factors that do not occur in synthetic formulae. This calls for novel complexity studies specific to feature models and dedicated methods to decompose this complexity in a form that makes it tractable for URS.

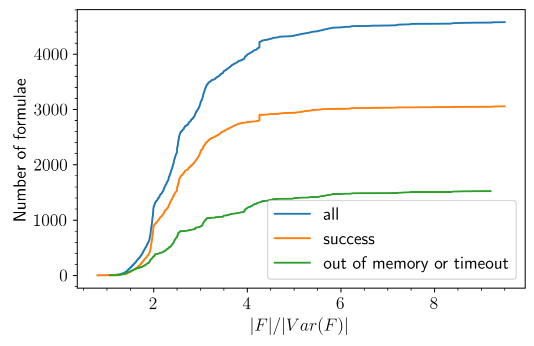


(a) SPUR

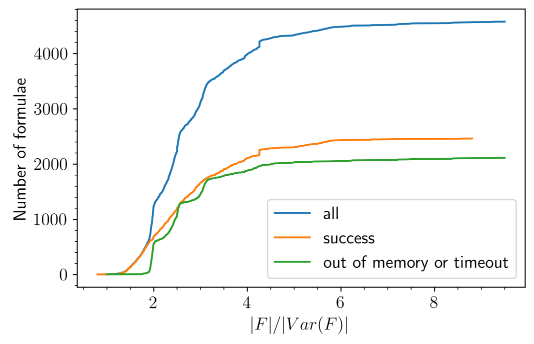


(b) UniGen3

Figure 8: Results on real-world formulae.



(a) SPUR



(b) UniGen3

Figure 9: CDF of real-world formulae wrt. $|F|/|Var(F)|$.

8 Threats to Validity

Internal Validity. This threat concerns the implementation and the choice of specific parameters during our experiments. We chose to execute our experiments on random formulae with 50 and 75 variables, which is less than most real-world formulae. This was necessary to execute this large synthetic benchmark on multiple solvers and avoid long executions and timeouts. Yet, our experiments showed that synthetic formulae are much more difficult to count or sample from than some real-world formulae with more variables. Regarding the community attachment model, we set the number of communities to 5. This may introduce a bias towards formulae with specific modularity. Thus, we reran the experiments with different numbers of communities and provided the results on our companion GitHub [50].

External Validity. There is no guarantee that our results generalize precisely to any formula and any model counter or sampler in each category. The reason behind this is the lack of general understanding of the complexity of SAT-based tasks [18], which we aim to address. To mitigate this threat, we selected a range of SAT formulae from multiple sources. They come from SAT Benchmarks used for the evaluation of uniform samplers [10, 11, 14] and various other sources such as feature models representing configurable systems of various types and sizes [1, 41, 47]. The datasets that we use include formulae that encode diverse types of models: electronic circuits [29], algorithmic problems, Linux kernels [41, 47], Unix command line tools, or configuration tools [24]. Thus, we are confident that our general conclusions are valid for a large class of real-world formulae.

9 Conclusion

In this paper, we analysed the experimental complexity of counting and uniform random sampling for Boolean formulae, under the prism of phase transitions and community structure. Our investigation initially focused on synthetic formulae, allowing us to finely explore phase transitions and the role of community structure in a controlled way. Hence, we demonstrated that phase transitions indeed occur in URS and #SAT, while the community structure impacts the amplitude of the peak. We also showed that phase transitions are harder to observe in real-world formulae, although many of these formulae remain intractable. We therefore believe that additional complexity factors are at play and need feature-model-specific studies to be explained. Overall, our work contributes to a principled understanding of URS and #SAT complexity, and we hope it can inspire future research in designing effective methods to approach these problems.

Acknowledgments

This research was funded in whole or in part by the Luxembourg National Research Fund (FNR). Gilles Perrouin is an FNRS Research Associate. Maxime Cordy and Olivier Zeyen are supported by FNR Luxembourg (grants INTER/FNRS/20/15077233/Scaling Up Variability/Cordy, C23/IS/18177547/VARIANCE, 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. doi:10.1145/3461002.3473070

[2] D. Achlioptas, Zayd Hammoudeh, and P. Theodoropoulos. 2018. Fast Sampling of Perfectly Uniform Satisfying Assignments. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 135–147.

[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. doi:10.1145/3491210

[4] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. 2012. The Community Structure of SAT Formulas. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 410–423.

[5] Roberto J Bayardo Jr and Joseph Daniel Pehoushek. 2000. Counting models using connected components. In *AAAI/IAAI*. 157–162.

[6] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. 2005. Automated Reasoning on Feature Models. In *Proceedings of CAiSE'05*. 491–503.

[7] Elazar Birnbaum and Eliezer L Lozinskii. 1999. The good old Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research* 10 (1999), 457–477.

[8] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. 2008. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering* 20 (2008), 172–188.

[9] Shaowei Cai, Chuan Luo, and Kaile Su. 2014. Scoring Functions Based on Second Level Score for k-SAT with Long Clauses. *J. Artif. Intell. Res.* 51 (2014), 413–441. https://api.semanticscholar.org/CorpusID:16385185

[10] 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.

[11] 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. doi:10.1145/2593069.2593097

[12] Sheng Chen and Martin Erwig. 2011. Optimizing the Product Derivation Process. *2011 15th International Software Product Line Conference* (2011), 35–44.

[13] Gilles Dequen and Olivier Dubois. 2006. An Efficient Approach to Solving Random k-SAT Problems. *Journal of Automated Reasoning* 37 (2006), 261–276. https://api.semanticscholar.org/CorpusID:25414864

[14] 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. doi:10.1145/3180155.3180248

[15] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *SAT'03*. Springer, 502–518.

[16] Guillaume Escamocher and Barry O'Sullivan. 2022. Generation and Prediction of Difficult Model Counting Instances. *ArXiv* abs/2212.02893 (2022).

[17] Johannes Klaus Fichte, Markus Hecher, and Florim Hamiti. 2020. The Model Counting Competition 2020. *Journal of Experimental Algorithmics (JEA)* 26 (2020), 1–26.

[18] 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.

[19] Jian Gao, Ruizhi Li, and Minghao Yin. 2017. A randomized diversification strategy for solving satisfiability problem with long clauses. *Science China Information Sciences* 60 (2017), 1–11. https://api.semanticscholar.org/CorpusID:13219850

[20] Jian Gao, Minghao Yin, and Ke Xu. 2011. Phase transitions in knowledge compilation: an experimental study. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 364–366.

[21] Ian P. Gent and Toby Walsh. 1994. The SAT Phase Transition. In *European Conference on Artificial Intelligence*, Vol. 94. PITMAN, 105–109.

[22] Jesús Giráldez-Cru and Jordi Levy. 2015. A Modularity-Based Random SAT Instances Generator. In *International Joint Conference on Artificial Intelligence*. AAAI Press.

[23] Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. 2020. Phase Transition Behavior in Knowledge Compilation. In *International Conference on Principles and Practice of Constraint Programming*. https://api.semanticscholar.org/CorpusID:220665810

[24] 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). doi:10.1007/s10664-018-9635-4

[25] Jinbo Huang and Adnan Darwiche. 2004. Using DPLL for efficient OBDD construction. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 157–172.

[26] 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.

[27] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1-2 (1938), 81–93.

[28] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. 2010. Model Counting in Product Configuration. In *LoCoCo*.

[29] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler. In *IJCAI*.

[30] 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. doi:10.1145/2791060.2791070

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

[32] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. 2009. SAT-based Analysis of Feature Models is Easy. In *Proceedings of the 13th International Software Product Line Conference* (San Francisco, California, USA) *(SPLC '09)*. Carnegie Mellon University, Pittsburgh, PA, USA, 231–240. http://dl.acm.org/citation.cfm?id=1753235.1753267

[33] 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.

[34] David G. Mitchell, Bart Selman, and Hector J. Levesque. 1992. Hard and Easy Distributions of SAT Problems. In *AAAI Conference on Artificial Intelligence*, Vol. 92. 459–465.

[35] 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.

[36] Zongxu Mu and Holger H. Hoos. 2015. On the Empirical Time Complexity of Random 3-SAT at the Phase Transition. In *International Joint Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:17251755

[37] Mark E. J. Newman and Michelle Girvan. 2003. Finding and evaluating community structure in networks. *Physical review. E, Statistical, nonlinear, and soft matter physics* 69 2 Pt 2 (2003), 026113.

[38] 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. doi:10.1145/3106237.3106273

[39] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Margaret Myers. 2020. *Scalable Uniform Sampling for Real-World Software Product Lines*. Technical Report TR-20-01.

[40] 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.

[41] 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.

[42] Yash Pote, Saurabh Joshi, and Kuldeep S. Meel. 2019. Phase Transition Behavior of Cardinality and XOR Constraints. *ArXiv* abs/1910.09755 (2019). https://api.semanticscholar.org/CorpusID:199465708

[43] Marko Samer and Stefan Szeider. 2007. Algorithms for propositional model counting. *J. Discrete Algorithms* 8 (2007), 50–64.

[44] Mate Soos. 2024. *Benchmarks used for Approximate Model Counting*. doi:10.5281/zenodo.10449477

[45] 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)*.

[46] Joshua Sprey, Chico Sundermann, Sebastian Krieter, Michael Nieke, Jacopo Mauro, Thomas Thüm, and Ina Schaefer. 2020. SMT-based variability analyses in FeatureIDE. *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems* (2020), 1–9.

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

[48] Chico Sundermann, Tobias Heß, Michael Nieke, Paul Maximilian Bittner, Jeffrey M. Young, Thomas Thüm, and Ina Schaefer. 2023. Evaluating state-of-the-art # SAT solvers on industrial configuration spaces. *Empirical Software Engineering* 28, 2 (2023), 29.

[49] 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.

[50] Olivier Zeyen. 2025. URS Phase Transitions. https://github.com/serval-uni-lu/urs_phase_transitions, https://doi.org/10.5281/zenodo.15807056.

[51] 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 *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE)*. 23–32.