Generating SAT Instances with Community Structure ☆

Jesús Giráldez-Cru

Artificial Intelligence Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, Spain

Jordi Levy

Artificial Intelligence Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, Spain

Abstract

Nowadays, modern SAT solvers are able to efficiently solve many *industrial*, or *real-world*, SAT instances. However, the process of development and testing of new SAT solving techniques is conditioned to the finite and reduced number of known industrial benchmarks. Therefore, new models of random SAT instances generation that capture realistically the features of real-world problems can be beneficial to the SAT community. In many works, the structure of industrial instances has been analyzed representing them as graphs and studying some of their properties, like *modularity*.

In this work, we use the notion of modularity to define a new model of generation of random SAT instances with community structure, called *Community Attachment*. For high values of modularity (i.e., clear community structure), we realistically model pseudo-industrial random SAT formulas. This model also generates SAT instances very similar to classical random formulas using a low value of modularity. We also prove that the phase transition point, if exists, is independent on the modularity. We evaluate the adequacy of this model to real industrial SAT problems in terms of SAT solvers performance, and show that modern solvers do actually exploit this community structure. Finally, we use this generator to observe the connections between the modularity of the instance and some components of the solver, such as the variable branching heuristics or the clause learning mechanism.

1. Introduction

The Boolean Satisfiability Problem (SAT) is one of the most studied problems in Computer Science. It is the first known NP-complete problem, as proved by S. Cook in 1971 [13]. However, many application problems are nowadays encoded into SAT instances, and efficiently solved by modern SAT solvers. Namely, these solvers are the so known Conflict-Driven Clause Learning (CDCL) SAT solvers. In the last years, these solvers have been the dominant *technique* in the SAT community to solve application problems, including a wide variety of domains, such as cryptography, hardware and software verification, planning, or scheduling, among others.

It is accepted that random and industrial SAT instances have a distinct nature. While random formulas can be easily generated on demand, the set of industrial benchmarks, which encode real-world problems, is limited. The problem of generating realistic pseudo-industrial random instances is stated in [37] as one of the most important *challenges* for the next few years: *"Challenge 10: Develop a generator for problem instances that have computational properties that are more similar to real world instances"*. This challenge is also stated in [25, 14]. The main motivation of this challenge is improving the process of development and testing of SAT solvers, and their possible specialization.

There exist a wide variety of works on the analysis of the nature of industrial SAT instances. The intuition is that these formulas have some kind of *structure*, which is exploited by CDCL SAT solvers. In many of these works,

☆ This work is partially supported by the CSIC project 201450E045.

Email addresses: jgiraldez@iiia.csic.es (Jesús Giráldez-Cru), levy@iiia.csic.es (Jordi Levy)

URL: http://www.iiia.csic.es/~jgiraldez (Jesús Giráldez-Cru), http://www.iiia.csic.es/~levy (Jordi Levy)

SAT formulas are represented as graphs, and some (graph) features are studied. The classical Erdös-Rényi model has been extensively used for generating random graphs. In this model, the degree of nodes follows a binomial distribution, with small variability. This is exactly the case of classical random formulas. However, the structure of most real-world networks cannot be described with this classical model, and therefore, new models have been defined. For instance, *Preferential Attachment* [10] is used to explain the scale-free structure of some real networks, where the nodes degree follows a power-law distribution, with big variability. *Similarity* has also been proposed as a second mechanism, that together with *popularity* or preferential attachment, may result into complex networks with an hyperbolic topology [36]. In [2] it is described an hyperbolic graph generator. When SAT instances are modeled as graphs, many graph properties can be analyzed, such as the small-world property [41], the scale-free structure [5], the eigenvector centrality [24], or the self-similarity [3]. Moreover, some notions of structure have been already used to better understand certain components of modern SAT solvers [27, 26, 4].

In this paper, we focus on the notion of *modularity* of a graph [32]. In a graph with high modularity (i.e., clear *community structure*), we can find a partition of its nodes into communities such that most of its edges connect nodes of the same community. In [7], it is analyzed the modularity of the industrial benchmarks used in SAT competitions. In that paper, it is shown that most industrial SAT formulas exhibit a high modularity. On the contrary, randomly generated instances have a very low modularity. The community structure has been shown correlated with the runtime of CDCL SAT solvers [34, 35]. Moreover, it has been used to improve the performance of some solvers [29, 40, 8, 31].

One important motivation for the development of *pseudo-industrial* SAT instances generators is to *isolate* some known properties of these real-world problems. This allows us to study the impact of these properties on the performance and behavior of SAT solvers. This approach has already been used in [6]. In that work, authors present a random SAT instances generator which takes into account the scale-free structure of real-world SAT instances to generate formulas in which the number of variable occurrences follows a power-law distribution. Using it, they observe that CDCL SAT solvers focus their decisions on the most frequent variables. In the case of community structure, similar questions also arise. For instance, do SAT solvers concentrate their decisions on variables of the same (or few) communities? Do the conflicts found by the solver relate variables of the same community? How does the activity of each community evolve along the execution of the search? Answering these questions may help to better understand the different ingredients of modern SAT solvers and their impact on the solving process, with the long-term aim of improving them.

The main contribution of this work is a new model of generation of random SAT instances based on the notion of modularity[1]. With this new model, we can generate formulas for any given value of modularity. For a high modularity, the resulting instance is more adequate to model industrial problems than classical random formulas. On the contrary, with a low modularity we can generate SAT instances very similar to classical random ones. We show that this model works appropriately for different input values of number of variables n and clauses m. We also show that, if there exists a phase transition point SAT-UNSAT when the ratio clause/variable is increased, then it does not depend on the modularity. We give empirical evidence that the performance of SAT solvers is consistent with the expected properties of the generated formulas, i.e. SAT solvers *specialized* in industrial problems perform better in high modular instances than SAT solvers *specialized* in random formulas, and vice versa. Finally, we use this generator to answer the questions stated in the previous paragraph. In particular, we analyze the relations between the community structure and the branching decision heuristics, and the mechanism of learning new clauses from the conflicts the solver finds along its execution. A preliminary version of this paper was published as [18].

The rest of the paper proceeds as follows. We first reference some related works on the generation of pseudo-industrial SAT instances in Section 2. After some preliminaries on modularity of graphs in Section 3, we describe the generation model in Section 4. In Section 5, we show that this model works appropriately for different input values. In Section 6, we analyze the phase transition point of the instances generated by our model. In Section 7, we show the adequacy between the performance of SAT solvers and the properties of the generated instances. In Section 8, we use this generator to analyze some components of a CDCL SAT solver. Finally, we conclude in Section 9 and propose some future work.

[1] This modularity-based random SAT instances generator is available in `http://www.iiia.csic.es/~jgiraldez/software`

2. Related Work

There already exist some works facing the problem of generating *pseudo-industrial* random instances. We distinguish between those works focused on particular problems domains from those others that generalize common properties of industrial instances.

In the case of particular problems domains, we have the following works. Gomes and Selman [19] proposed a new model of SAT benchmarks resulting from introducing random perturbations into structured problems. These benchmarks encode instances of the partial latin square completion problem, with some randomly selected pre-assigned values. Achlioptas et al. [1] refined the previous model to guarantee the generation of satisfiable instances. Gent et al. [16] introduced a method, called *morphing*, for introducing randomness into structured problems, resulting into problems with small-world topology. Järvisalo et al. [22] proposed an instance generator based on finding optimal circuits of Boolean functions. In general, these methods do not explicitly generate SAT instances with community structure. However, this feature may appear for certain input parameters.

On the other hand, other works focus on general properties shared by the majority of real-world problems. Burg et al. [12] presented a generator that combines subparts of real-world instances to create new ones. Notice that this method may generate instances with community structure when the input formulas have such structure. Nevertheless, the modularity of the generated instances cannot be controlled. Ansótegui et al. [6] used the notion of scale-free graph to generate formulas where the number of variable occurrences follows a power-law distribution. This feature is very characteristic in real-world instances, but it is not a sufficient condition to guarantee community structure. Newsham et al. [34] proposed a random 3-CNF generator which divides the set of variables into groups, and uses a certain probability to decide for each clause whether a literal is selected from the same group than the previous one or not (randomly selected in the case of the first literal of the clause). This way, a high value in this probability generates very modular instances. Slater [39] proposed a method to generate SAT instances with *small-world* topology. This feature is also very common in many real-world networks, and it is characterized by a small typical distance and a high clustering coefficient. The proposed model is characterized by n variables, m clauses, c clusters and a parameter p. They generate c independent 3-SAT formulas, each one having n/c variables and $(1-p)\,m/c$ clauses, and then add $p\,m$ link clauses using the entire set of variables, in the spirit of the techniques used traditionally to generate small-world graphs. This model is quite similar to ours, and may also result into instances with high modularity. Notice, however, that this paper was written in 2002, and the notion of *modularity* was defined in 2004. The main difference is that the relation between the modularity of their generated instances and the parameter p is not so clear as in our model. In particular, we force link clauses to connect variables of *distinct* communities, and we do not require communities to have the same number of internal clauses. This way, we show the relation in our model between the modularity and the parameter p. This relation lets us to control the modularity of the generated instances, which has important effects on the performance of the solver (as shown in [34, 35]).

3. Preliminaries

SAT is the problem of deciding if the variables of a propositional formula can be assigned in such a way that the formula is evaluated as `true`. A *literal* is either a variable or its negation, a *clause* is a disjunction of literals, a conjunctive normal form (*CNF*) formula is a conjunction of clauses, and a k-*CNF* is a CNF in which all clauses have exactly k literals.

A weighted graph G is a pair $G = (V, w)$, where V is the set of nodes, and $w : V \times V \to \mathbb{R}^+$ is the edge-weight function that satisfies $w(x, y) = w(y, x)$.

The Variable Incidence Graph (VIG) of a SAT instance Γ is the graph whose nodes represent the variables of Γ, and there exists an edge between two variables if they both appear in a clause C. A clause with $|C|$ literals results into $\binom{|C|}{2}$ edges. Thus, to give the same relevance to all clauses, we assign to edges the weight $w(x, y) = \sum_{\substack{C \in \Gamma \\ x,y \in C}} 1/\binom{|C|}{2}$.

The *community structure* of a graph is usually measured using the notion of *modularity* [33]. Defined for a graph G and a partition P of its vertexes into communities, the modularity Q (see Eq. 1) measures the fraction of internal edges (edges connecting vertexes of the same community) w.r.t. a random graph with same number of vertexes and same degree. This avoids that the best partition is the one made up by an only community containing all vertexes.

Algorithm 1: Community Attachment

Input: **int** n, m, c, k; **real** Q;
Output: k-CNF **SAT Instance** Γ

$\Gamma := \emptyset$;
$p := Q + 1/c$;
for $j \in \{1, \ldots, m\}$ **do**
  // Select the community c_i of each literal
  if $rand([0,1]) \leq p$ **then** // all literals in the same community
    $r := rand(\{1, \ldots, c\})$;
    for $i \in \{1, \ldots, k\}$ **do**
      $c_i := r$;
  else // all literals in distinct communities
    for $i \in \{1, \ldots, k\}$ **do**
      repeat
        $c_i := rand(\{1, \ldots, c\})$;
      until $\forall i' < i (c_{i'} \neq c_i)$;
  // Create the clause C
  $C := \emptyset$;
  for $i \in \{1, \ldots, k\}$ **do**
    repeat
      $x_i := rand(\{\lfloor (c_i - 1)\, \frac{n}{c} \rfloor + 1, \ldots, \lfloor c_i\, \frac{n}{c} \rfloor\})$; // x_i random var of community c_i
    until $\forall i' < i (x_{i'} \neq x_i)$; // Avoid trivial clauses;
    $C := C \bigvee rand(\{-1, 1\}) \cdot x_i$; // random polarity
  $\Gamma := \Gamma \bigwedge C$;
return Γ;

$$Q(G,P) = \sum_{P_i \in P} \frac{\sum_{x,y \in P_i} w(x,y)}{\sum_{x,y \in V} w(x,y)} - \left(\frac{\sum_{x \in P_i} deg(x))}{\sum_{x \in V} deg(x)} \right)^2 \tag{1}$$

The modularity of a graph is the maximal modularity for any possible partition: $Q(G) = \max\{Q(G,P) \mid P\}$.

Computing the modularity of a graph is NP-hard [11]. Due to its complexity, instead of computing the (exact) modularity, most of methods in the literature approximate a lower-bound in the value of Q.

4. Community Attachment Model

In the classical random k-SAT model, a random formula $F_k(n,m)$ is a set of m clauses over n variables, where clauses are chosen uniformly and independently among all $2^k \binom{n}{k}$ non-trivial clauses of length k. A non-trivial clause of length k contains k distinct, non-complementary literals. In this paper we present a new model of random formulas: the **Community Attachment** (**CA**) model. This is parametric in a probability p and a number of communities c of the set of variables. It allows the generation of formulas of any desired modularity.

Definition 1. Community Attachment. *Let N be a set of n variables, a partition P of N into c pairwise disjoint communities of the same size n/c, with $k \leq c \leq n/k$, and a real value $0 \leq p \leq 1$. A random formula $F_k(n,m,c,p)$ is a set of m non-trivial clauses with k literals over the n variables, selected independently as follows. With probability p, choose a clause uniformly among all $c\, 2^k \binom{n/c}{k}$ clauses with all literals in the same community; and with probability $1-p$, a clause uniformly among all $\left(\frac{2\,n}{c}\right)^k \binom{c}{k}$ clauses with all literals in distinct communities.*

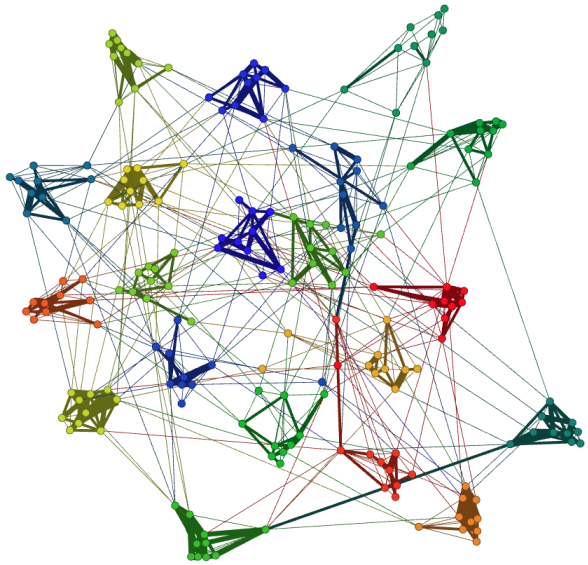

Figure 1: Variable Incidence Graph (VIG) of a random instance from $F_k(n, m, c, p)$ generated with $n = 200$ variables, $m = 425$ clauses of length $k = 3$, $c = 40$ communities, and modularity $Q = 0.8$ ($p = Q + 1/c$). Variables (nodes) of the same community are plotted with the same color. Edges are scaled according to their weight.

Notice that in the previous definition we need to impose the restriction $k \leq c \leq n/k$ to ensure that there always exists at least one possible clause to select. Notice also that for $k = 2$ and $p = \frac{n/c-1}{n-1}$ we have the classical 2-SAT model: $F_2(n, m) = F_2(n, m, c, \frac{n/c-1}{n-1})$, but for bigger k the Community Attachment model does not subsume the classical model.

Given a SAT formula Γ with n variables and m clauses, consider the VIG G of Γ. Our model ensures a lower-bound for the modularity of this graph.

Theorem 2. *Given a formula $\Gamma \in F_k(n, m, c, p)$, let G be its VIG. The average modularity of G is bounded as:*

$$E[Q(G)] \geq p - \frac{1}{c}$$

PROOF: Recall that modularity is defined as the maximal modularity for all possible partitions of the nodes into communities. Here we consider the partition used to generate the formula. For this particular partition P, when we select a clause with all variables in the same community (with probability p), we get $\binom{k}{2}$ internal edges. The sum of the weights of the edges generated by a single clause is always 1. Therefore, the fraction of internal edges is, on average, $\frac{p\,m}{m}$. The sum of nodes degrees is $2m$, thus $2m/n$ is the expected node degree. Since n/c is the number of nodes per community, the sum of nodes degrees in one community is on average $\frac{n}{c}\frac{2m}{n}$.

Summarizing, for this partition P, we get

$$E[Q(G, P)] = \frac{p\,m}{m} - c\left(\frac{\frac{n}{c}\frac{2m}{n}}{n\frac{2m}{n}}\right)^2 = p - \frac{1}{c}$$

that is a lower-bound for the expected modularity $E[Q(G)]$. ■

When p close to 1, the expected modularity $E[Q(G)]$ is very close to this lower-bound $p - 1/c$, because the partition used in the formula generation is very similar to the optimal. Therefore, we can use the previous theorem to

generate formulas with a desired modularity Q. We simply take:

$$p = Q + \frac{1}{c} \tag{2}$$

which ensures at least a modularity Q. In practice, as we will see in Section 5, the formulas we obtain have a modularity $Q \approx p - 1/c$, except when p and m/n are small.

Notice that the previous lower-bound of the modularity depends of the graph model used to represent the formula. For instance, the CVIG model [8] uses a bi-partite structure to represent both variables and clauses into nodes, and puts an edge between a variable-node and a clause-node when that variable appears in that clause. If, instead of the VIG model, we use the CVIG G'[2], we can get a lower-bound $E[Q(G')] \geq p - p/c$. Notice, however, that in the case we use the definition of modularity for bi-partite graphs, which is slightly different. Nevertheless, both approximations are very similar. Since most of the methods in the literature to compute the community structure are efficient for non bi-partite graphs (as the VIG model), we use the approximation of Theorem 2.

In Algorithm 1, there is an implementation of the Community Attachment random formulas generator from $F_k(n, m, c, p)$. Using $p = Q + 1/c$ these formulas will have an expected modularity close to Q.

In Figure 1, we represent the VIG of an instance generated with $n = 200$ variables, $m = 425$ clauses, modularity $Q = 0.8$ and $c = 40$ communities. In this plot, variables (nodes) of the same community are plotted with the same color, and edges are scaled according to their weight. As the value of the modularity is high, it is more likely that a clause relates variables of the same community. Therefore, the weight of the edges connecting nodes (variables) of the same community is higher, as expected.

5. Validation of the Model

In order to analyze the community structure of the SAT instances obtained with our model, we have generated some sets of random formulas for different values of $Q \in \{0.9, 0.8, 0.7, 0.5, 0.3\}$ (hence $p = Q + 1/c$), and for different values of number of communities $c \in \{10, 20, 40, 80\}$. Each set contains 50 random instances. Remark that the modularity Q of (real) industrial SAT instances is usually greater than 0.7 [7], while no modularity greater than 0.3 is found for classical random k-CNF formulas. Moreover, the number of communities c is usually in the interval $[10, 100]$ [7].

In Figure 2, we analyze their modularity Q' (top) and their number of communities c' (bottom), varying the number of variables n, for a fixed clause/variable ratio $m/n = 4$ (left), and varying the clause/variable ratio m/n, for a fixed number of variables $n = 1000$ (right). In this experiment, the number of communities c is fixed to $c = 40$. Notice that the main goal of this experiment is to show that our model generates instances with modularity Q' and number of communities c' similar to the input parameters Q and c, for any number of variables n and clauses m. We use the algorithm described in [7] to compute an approximation of Q' and c'. In fact, this algorithm computes a lower-bound of the modularity. The dispersions of the approximated Q' and c' are very small, so they are not shown in the plots.

We observe that the modularity Q' and the number of communities c' are almost unaffected by these variations of n and m/n. In general, the approximation computed for Q' is slightly smaller than expected, and the partition into communities is also very similar to the partition used in the generation. For small values of the clause/variable ratio m/n and the probability p, the number of clauses relating variables of the same expected community is very small. This produces the existence of some unconnected sub-communities within each expected community. Hence, Q' and c' are much greater than expected, and Q' cannot be estimated as $p - 1/c$. When we generate formulas with small values of p, e.g. $Q = 0.3$ and $c = 40$, we observe that, although the formulas have a guaranteed lower-bound of $Q \geq 0.3$, the computed approximation of Q' is smaller (close to 0.2 when $n \approx 20000$). The number of communities c' is also smaller than the expected c. In this case, this error is not produced by our model. It is due to the greediness of the algorithm [7] used to approximate Q', which is not able to find a similar partition to the one used in the generation.

In a second experiment, we generate families of instances with the same number of variables n and the same clause/variable ratios m/n than in the previous experiment, and varying the number of communities c with a fixed

[2]We use a partition with all variable-nodes and all internal clause-nodes in their corresponding community, and an extra community with all external clause-nodes.

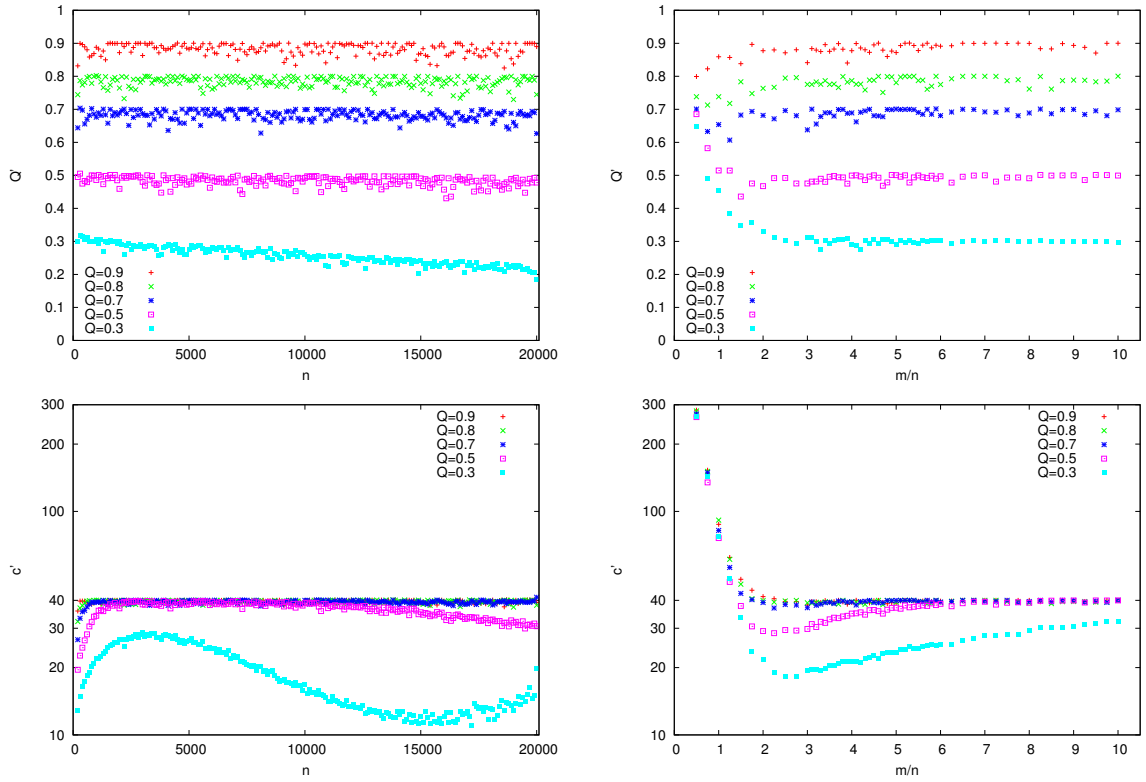

Figure 2: Approximations of modularity Q' (top) and number of communities c' (bottom) of some sets of random SAT formulas from $F_k(n, m, c, p)$, varying the number of variables n with $m/n = 4$ (left), and varying the clause/variable ratio m/n with $n = 1000$ (right), with $k = 3$, $c = 40$, and $p = Q + 1/c$. Each plotted data is the average of 50 instances.

modularity $Q = 0.8$. In Figure 3, we represent the results. We observe that the expected modularity Q' as well as the expected number of communities c' is almost unaffected in these families, except, as expected, for small values of n and m/n. In fact, when the value of the number of communities c is high enough (as actual industrial SAT instances have), its relevance in our model is very small, as expressed by Equation 2. Hereinafter we use in our experiments a fixed value of number of communities $c = 40$, assuming that this value is representative for real application problems, and therefore without altering general conclusions observed in the empirical results.

6. Phase Transition

In classical random k-CNF instances, some interesting properties, as the satisfiability or the hardness, are correlated to the clause/variable ratio m/n [30]. The Satisfiability Threshold Conjecture, which remains open for $k > 2$, suggests that it may exist a critical ratio r, such that below this point all formulas from $F_k(n, m)$ are SAT (under-constrained) and above it they all are UNSAT (over-constrained) with uniformly positive probability, when n tends to infinity. Experimentally, this phase transition point has been shown to be around $r \approx 4.26$ for $k = 3$. Moreover, the hardness of these instances is also characterized by this parameter: closer to this ratio, harder the instance.

In this section we check if this phenomenon also exists in the random SAT instances generated with our model, and if the new transition point, noted r', differs from the classical $r' \neq r$. In Figure 4, we represent the fraction of UNSAT instances for some sets of random formulas with distinct Q, varying the clause/variable ratio m/n. We observe that the fraction of UNSAT formulas increases with m/n. Therefore, for small (big) values of m/n, nearly all formulas are SAT (UNSAT). When Q is small, the value r' is close to the classical $r \approx 4.26$. Recall that when $p \approx 1/c$, our model is quite similar to the classical random k-SAT model. However, we also observe that, when Q increases, r' decreases.

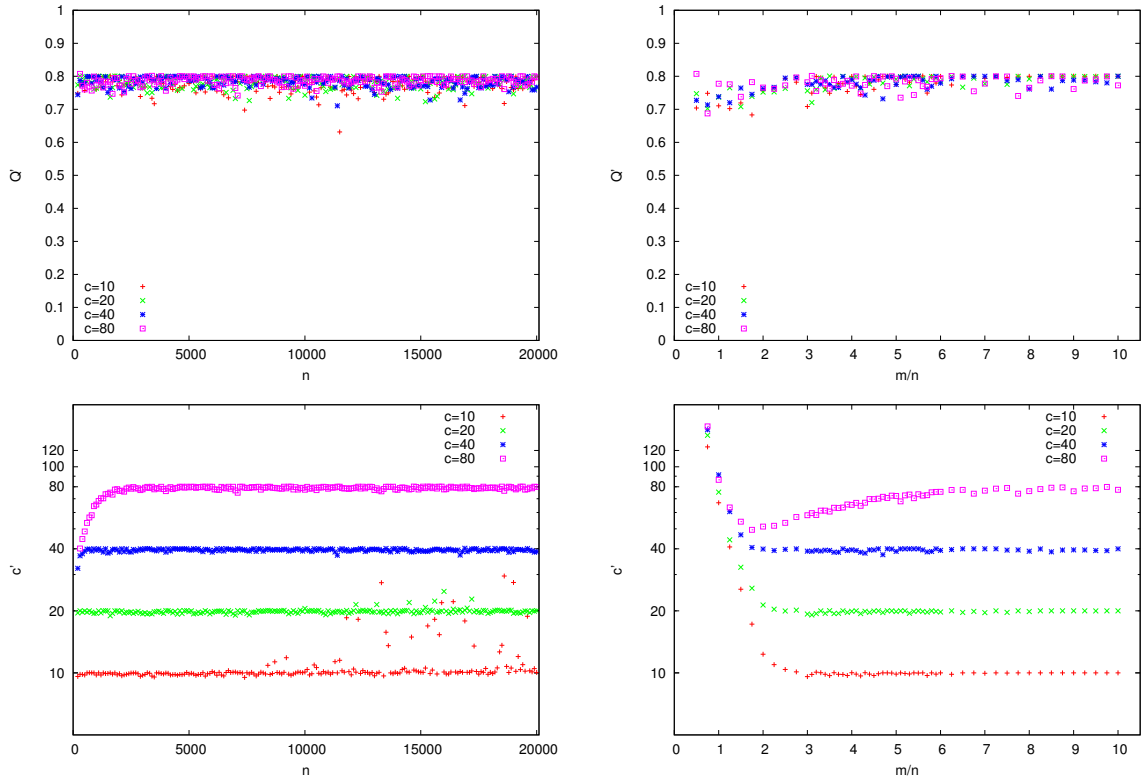

Figure 3: Approximations of modularity Q' (top) and number of communities c' (bottom) of some sets of random SAT formulas from $F_k(n, m, c, p)$, varying the number of variables n with $m/n = 4$ (left), and varying the clause/variable ratio m/n with $n = 1000$ (right), with $k = 3$, $Q = 0.8$, and $p = Q + 1/c$. Each plotted data is the average of 50 instances.

In our experimentation, for each family of instances we use the biggest value of number of variables n allowing us to get a solution in less than 3 hours. Therefore, this value may change for each family of instances. In Table 1 we report the phase transition point r' we found for some families of formulas with $k = 3$, varying the modularity Q. We also report the solver we used to solve the formulas, as well as the average and standard deviation of the runtime used by this solver. Remark that as the number of variables n of each family is different, their runtimes cannot be used to compare their hardness. We observe that the phase transition point r' decreases as the modularity Q increases. In Table 2, we report the phase transition point r' of some families of instances with $k = 4$. Again, the phase transition point r' of these families tends to decrease as the modularity Q increases. Notice that the generated formulas with a high modularity are hard, compared to industrial formulas. This is because industrial instances have other properties, like the scale-free structure, that contribute to make them easier.

The natural question is if this decrease in r' is also valid for n tending to infinity. In order to explain this decrease in the phase transition point r', and predict the behavior when n tends to infinity, we will consider the extreme case with $p = 1$. In these formulas, clauses only contain variables of the same community. Therefore, the formula is composed by c unconnected sub-formulas, and the whole formula is UNSAT if, and only if, at least one of the sub-formulas is UNSAT. Moreover, in this extreme case, all sub-formulas follow the classical model $F_k(n/c, m')$, for some m'. On average, all sub-formulas contain $E[m'] = m/c$ clauses; and all of them contain n/c variables. Hence, the average clause/variable fraction in sub-formulas is also $E[\frac{m'}{n/c}] = \frac{m/c}{n/c} = m/n$. However, even when the fraction m/n is smaller than the classical r (and so the expected clause/variable ratio of the formula), with some probability, some of the sub-formulas may get a large portion of clauses m' such that $\frac{m'}{n/c} > r$. This makes that sub-formula UNSAT with high probability. This has the effect of decreasing the phase transition point for finite n and c.

When n/c tends to infinity, the situation is completely different as the following theorem states.

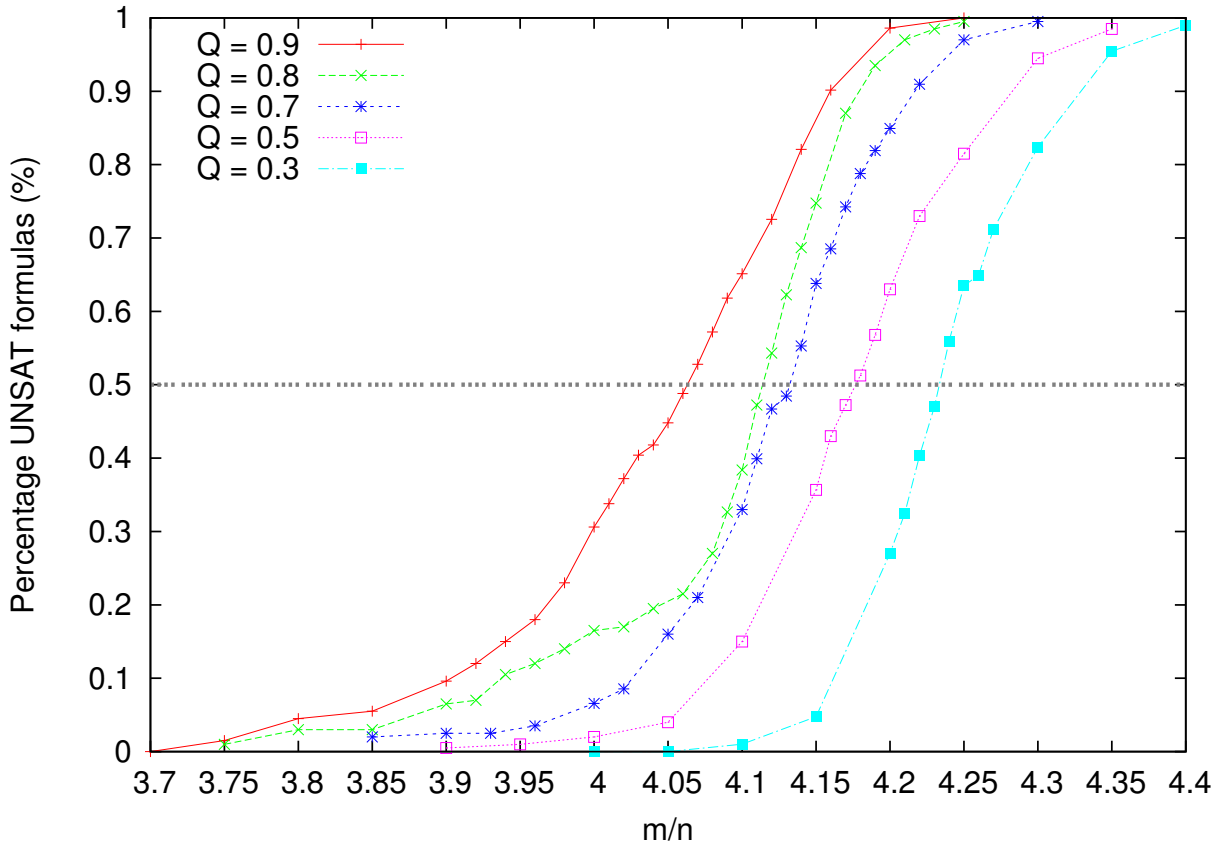

Figure 4: Fraction of UNSAT formulas for some sets of 200 random SAT formulas from $F_k(n, m, c, p)$, with $k = 3$, $c = 40$ and $p = Q + 1/c$, and varying the clause/variable ratio m/n (see the number of variables n of each family in Table 1).

Theorem 3. *The set of formulas $F_k(n, m, c, p)$, with $p = 1$ and any value of c satisfying $n/c \to \infty$, has a phase transition point r' at the same clause/variable ratio r of the classical formulas $F_k(n, m)$.*

PROOF: Let $r' = m/n$ and r be the classical phase transition point. The minimal r' such that the probability that some of the sub-formulas has more than $r\, n/c$ clauses (hence it is UNSAT with high probability when n/c tends to infinity), will be the phase transition point for this special case $p = 1$ of our model.

The probability that a given community P_i contains $r\, n/c$ clauses, when the formulas has m clauses, is

$$P(P_i \text{ is UNSAT}) = \frac{\binom{m}{r\, n/c}\, (c-1)^{m - r\, n/c}}{c^m}$$

Let $m = r'\, n$ be the number of clauses of the whole formula.

There are two cases:

First, we assume that the number of communities c tends to ∞ (but slower than n, hence n/c also tends to ∞).

Q	r'	n	solver	$\bar{R}$	$S\,[R]$
0.9	**4.06**	5000	Glucose	80.86	125.28
0.8	**4.11**	2000	Glucose	291.87	1217.23
0.7	**4.13**	1200	Glucose	211.64	791.76
0.5	**4.18**	600	March	544.19	1051.81
0.3	**4.24**	600	March	3492.36	3117.23

Table 1: Phase transition point r' of some sets of 200 random SAT instances from $F_k(n, m, c, p)$, with $k = 3$, $c = 40$ and varying Q. We also report the number of variables n, the solver and runtime R (average $\bar{R}$ and standard deviation $S\,[R]$) needed to solve them.

Q	r'	n	solver	$\bar{R}$	$S\,[R]$
0.9	**9.37**	2000	Glucose	10.14	10.45
0.8	**9.50**	1000	Glucose	2244.24	4122.35
0.7	**9.55**	600	Glucose	2239.89	3649.96
0.5	**9.60**	250	March	1524.22	1550.52
0.3	**9.79**	200	March	566.83	581.20

Table 2: Phase transition point r' of some sets of 20 random SAT instances from $F_k(n, m, c, p)$, with $k = 4$, $c = 40$ and varying Q. We also report the number of variables n, the solver and runtime R (average $\bar{R}$ and standard deviation $S\,[R]$) needed to solve them.

When $m, n \to \infty$, and $m/n \to \infty$, the binomials $\binom{m}{n}$, may be approximated as:

$$\begin{aligned}\binom{m}{n} = & \frac{m\cdot(m-1)\cdots(m-n+1)}{n!} \approx \frac{(m-n/2)^n}{\sqrt{2\pi n}(n/e)^n} = \\ = & \frac{m^n\left(1-\frac{1}{2m/n}\right)^{\frac{2m}{n}\cdot\frac{n^2}{2m}}}{\sqrt{2\pi n}(n/e)^n} \approx \frac{m^n(1/e)^{n^2/2m}}{\sqrt{2\pi n}(n/e)^n} = \\ = & \frac{1}{\sqrt{2\pi n}}\left(\frac{m\,e}{n\,e^{n/2m}}\right)^n\end{aligned}$$

using the middle value in the numerator, and the Stirling approximation in the denominator.

When $c \to \infty$, we can also approximate

$$(c-1)^{m-r\,n/c} = c^{m-r\,n/c}\,(1-1/c)^{c\frac{m-r\,n/c}{c}} \approx \frac{c^{m-r\,n/c}}{e^{\frac{m-rn/c}{c}}}$$

Replacing these two approximations, and $m = r'\,n$ we get

$$P(P_i \text{ is UNSAT}) \approx \frac{1}{\sqrt{2\pi rn/c}}\left(\frac{r'}{r}\exp\left(1-\frac{r'}{r}+\frac{1}{c}(1-\frac{r}{2r'})\right)\right)^{r\,n/c}$$

For $n/c, c \to \infty$, this function is dominated by the exponential factor

$$P(P_i \text{ is UNSAT}) = \mathcal{O}\left(\left(\frac{r'}{r}\exp\left(1-\frac{r'}{r}\right)\right)^{r\,n/c}\right)$$

The base of the exponentiation is strictly smaller than one except for $r = r'$. Therefore, when the number of communities and their size both tend to infinity, even in the extreme case $p = 1$, the probability that the formula is UNSAT is zero, for $r' < r$, i.e. the phase transition point is the same as for the classical random formulas.

In the second case, we assume that c is finite. Then, the approximation we have used for the binomial is not correct. When k is constant and $n \to \infty$, we may use

$$\binom{kn}{n} \approx \frac{1}{\sqrt{2\pi n\frac{k-1}{k}}}\left(\frac{k^k}{(k-1)^{k-1}}\right)^n$$

In this case we get

$$P(P_i \text{ is UNSAT}) = \mathcal{O}\left(\left(\frac{(\frac{r'}{r}c)^{\frac{r'}{r}c}}{(\frac{r'}{r}c-1)^{\frac{r'}{r}c-1}}\frac{(c-1)^{\frac{r'}{r}c-1}}{c^{\frac{r'}{r}c}}\right)^{r\,n/c}\right)$$

As in the previous case, the base of the exponentiation is one only when $r' = r$. Therefore, the phase transition point is also just the same as for classical random formulas. ■

In the classical model, we recall that the phase transition point, if exists, is only applicable when the size of the formula goes to infinity. In the context of SAT instances, the phase transition point r is, by definition, the clause/variable ratio such that *all* formulas below (equivalently, above) such ratio are satisfiable (equiv., unsatisfiable). In other words, the probability that an instance is UNSAT (equiv., SAT) has a value tending to 0 (equiv., 1) for any clause/variable ratio $r-\epsilon$ (equiv. $r+\epsilon$), for any $\epsilon > 0$. This abrupt change in the probability is represented as a vertical line in the point r, and this would only occur in extremely large instances, as suggested in the Satisfiability Threshold Conjecture. In the case of formulas of finite size, we observe that this abrupt change does not exist. In particular, we find an interval $(r-\epsilon, r+\epsilon)$ where the probability that an instance is UNSAT smoothly goes from 0 to 1. Empirical observations show that this change is more abrupt as the size of the formula grows, but no phase transition point has been found to date.

In the case of the Community Attachment model, we observe the same behavior. Therefore, both empirical observations of Figure 4 and the formal proof expressed in Theorem 3 match with the behavior expected from the classical model.

7. SAT Solvers Performance

In this section we show that *industrial-specialized* SAT solvers exploit the community structure of the formula, whereas *random-specialized* solvers do not.

In Figure 5 we compare the performance of the SAT solvers Glucose [9] (version 3.0) and March [20] (version br) over some sets of SAT formulas generated with our model, with distinct modularity values. While Glucose is a CDCL SAT solver which has been shown very good for solving industrial problems, March is a Look-ahead SAT solver commonly used to solve random k-CNF instances. We use sets of instances from $F_k(n, m, c, p)$ with a clause/variable ratio m/n in the phase transition point, a number of communities $c = 40$ and a clause length $k = 3$. We adjust the number of variables as in Table 1, in order to ensure that some of these solvers solve all formulas in a timeout of 3 hours.

We observe that, for high modularities (see $Q = 0.9$), Glucose solves all the instances, but March is only able to solve few UNSAT instances. More precisely, they are the ones in which there exists a very small unsatisfiability core, composed of variables of one or few communities. Notice that higher the modularity, more likely to find these instances with small refutations. It is also interesting to remark that Glucose also solves UNSAT formulas faster than SAT formulas when their modularity is high. As Q decreases, March is able to solve more instances (see $Q = 0.7$), and it starts to be as fast as Glucose, if it is not faster, when the modularity is small enough (see $Q = 0.5$). Finally, when Q is very small (see $Q = 0.3$), March is able to solve all the instances but Glucose only solves few of them. Remark that the number of variables is not the same for every family. We can conclude that a high modularity makes formulas easier to be solved by CDCL SAT solvers.

In Figure 6 we compare the performance of these two solvers with the instances of Table 2, i.e. $k = 4$. Again, we observe that high modular formulas (see $Q = 0.9$) are easy for Glucose, but March is only able to solve those of them having a small unsatisfiability core. As the modularity decreases, both solvers solve all instances, but March is still some orders of magnitude slower (see $Q = 0.5$). Finally, when the modularity is small, March shows a better performance than Glucose (see $Q = 0.3$).

These experiments suggest that the performance of the solver is affected by the structure of the formula (e.g., its community structure) independently of the sizes of the clauses.

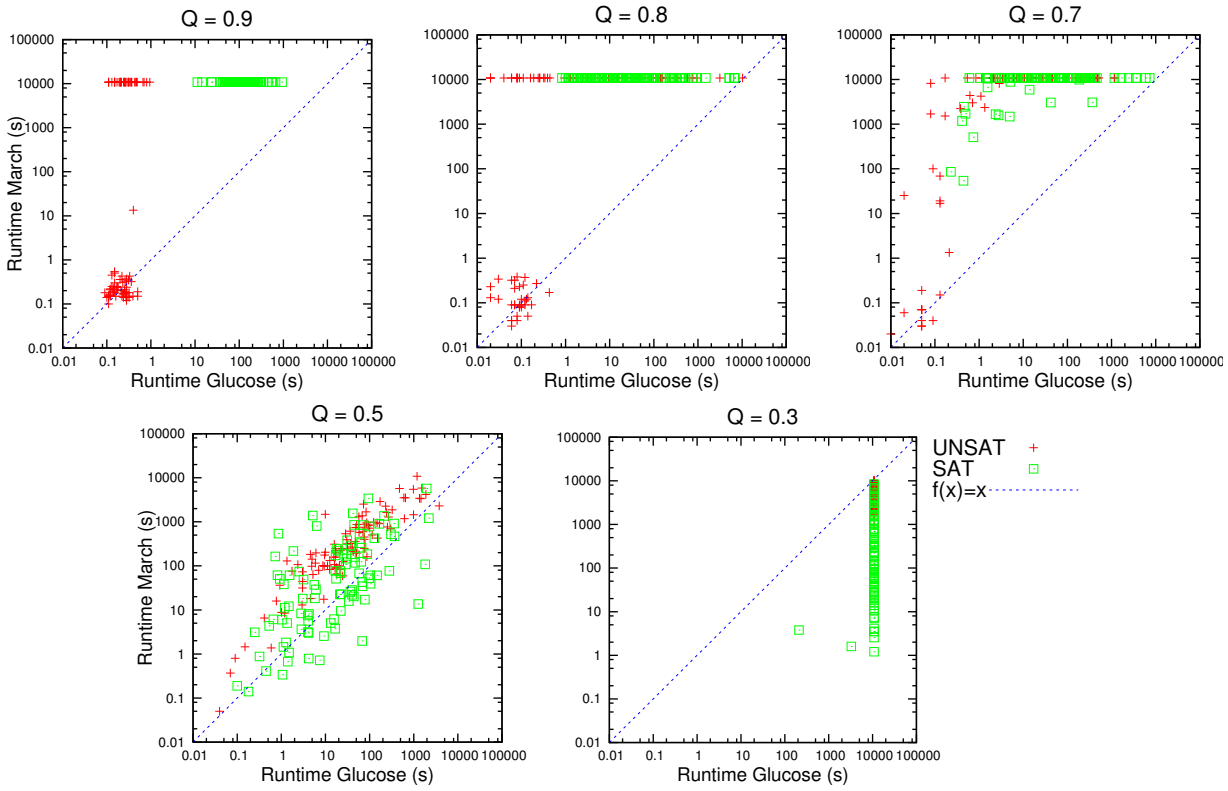

Figure 5: Relation between the runtimes (seconds) of Glucose and March, for some sets of 200 random SAT instances with $Q \in \{0.9, 0.8, 0.7, 0.5, 0.3\}$, $k = 3$ and $c = 40$ at the phase transition point (i.e., using families of Table 1). The timeout is set to 3 hours.

8. Analyzing the Components of a CDCL SAT Solver

One of the most important motivations for the development of this generator is to better understand the connections between the community structure of a SAT instance and the SAT solver components, with the long-term aim of improving them. In this section, we use our generator to study two main components of a CDCL SAT solver: the branching selection heuristics, and the conflict analysis and clause learning mechanism. Notice that these components are related. For instance, the activity of the variables participating in a conflict is increased, and the most active (and unassigned) variable is selected as the next decision variable. Even though, it seems convenient to study these components separately.

In this section, we use MiniSAT [15] (version 2.2) as a representative CDCL SAT solver. This is a very well known SAT solver on which many other SAT solvers, as Glucose, are based. Being a less sophisticated SAT solver allows us to analyze with more precision the impact of certain CDCL techniques on the community structure, without possible *noises* of other components. For instance, MiniSAT uses the Luby series to determine the number of conflicts between two restarts whereas Glucose uses a dynamic LBD-based strategy for this purpose. Therefore, while one can know when restarts are performed by MiniSAT, this is unknown *a priori* in Glucose. Also, we use a random SAT instance of the family from our model with $n = 1000$ variables, $m = 4200$ clauses, clause length $k = 3$, modularity $Q = 0.8$ and $c = 10$ communities. In this case, we use this reduced number of communities to improve the visualizations of results. However, similar results can be obtained with other typical values (e.g., $c = 40$). We remark that, even when we only plot results for a single instance, similar behaviors are observed in all instance of the family. Therefore, the conclusions drawn in this section are *general*. Recall that our generator, for simplicity, assigns n/c consecutive variables to each community. For instance, community c_1 contains variables with indexes from 1 to 100 (both communities and variables are 1-based numbered). MiniSAT solved this instances in 1.9 seconds, taking

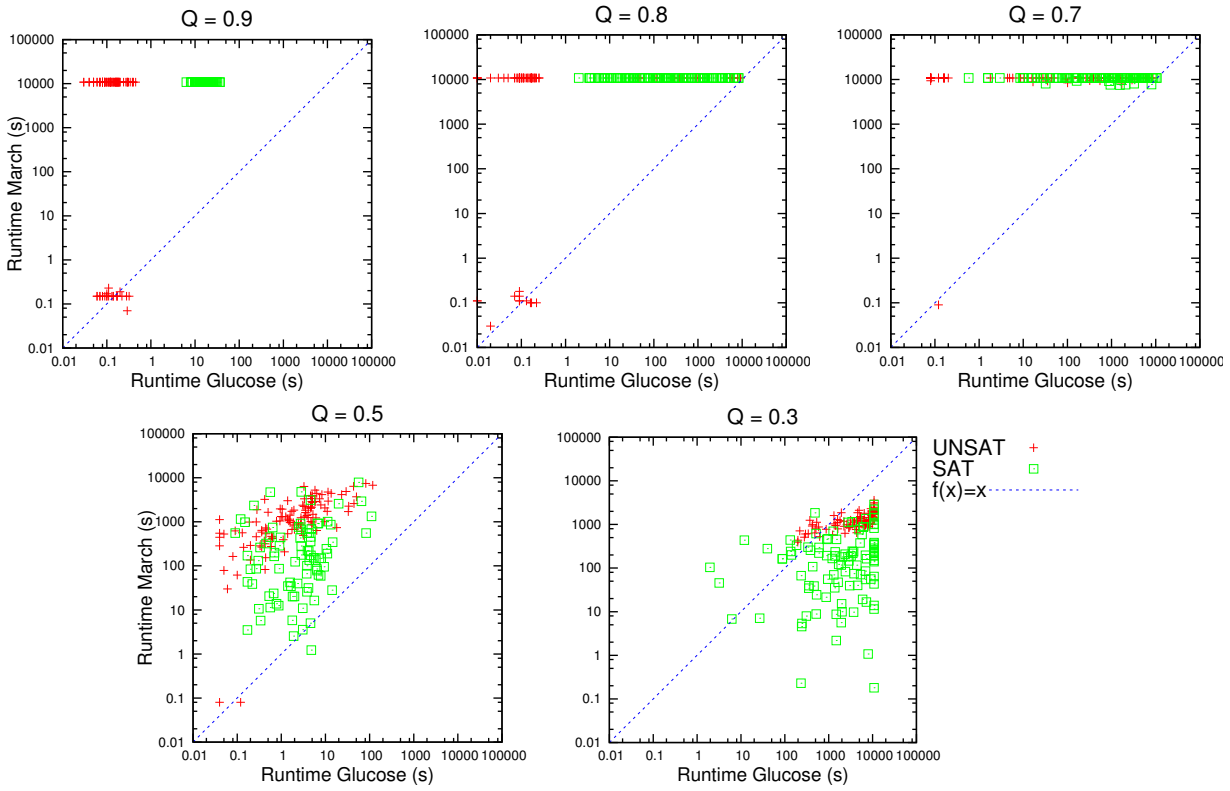

Figure 6: Relation between the runtimes (seconds) of Glucose and March, for some sets of 200 random SAT instances with $Q \in \{0.9, 0.8, 0.7, 0.5, 0.3\}$, $k = 4$ and $c = 40$ at the phase transition point (i.e., using families of Table 2). The timeout is set to 3 hours.

18226 decisions and finding 11722 conflicts. In Figures 7 and 8, the X-axis represents the number of conflict, and the Y-axis the index of variables. For clarity, we only plot the first 2000 conflicts. There are horizontal lines to split the set of variables belonging to each community, and vertical lines to represent the restarts along the execution.

First, we want to know if SAT solvers concentrate their decisions on variables of the same (of few) communities along their execution. In Figure 7 (top) we represent which variables are used to branch, i.e., the decision variables. As the X-axis represents the number of conflicts, the Y-axis shows the set of variables decided between two consecutive conflicts. We observe that the solver tends to focus its decisions on variables of the same community, during a period of time. After a while (when all variables of this community are assigned), it changes to another community. This behavior is repeated during the whole execution. The time the solver stays deciding in the same community is indeterminate, and does not depend on the restarts.

Second, in Figure 7 (center) we analyze the set of assigned variables. Notice that this set contains, not only decision variables, but also implied variables. We observe that all variables of the community where the solver has been focusing get assigned, and remain assigned when the solver decides to change to another community. In the next restart, all assigned variables in modules where the solver is not focused, get unassigned. For example, after 400 conflicts, a restart occurs. Before this restart, the solver was focused on community c_9, but all variables of communities c_3 and c_5 were also assigned (because the solver had also been focusing on these communities before c_9). After the restart, only variables of c_9 are assigned again. Therefore, restarting policy has the effect of reinforcing the focus of assigned variables on the same community.

Finally, we want to study if the conflicts found by the solver relate variables of the same community. In Figure 7 (bottom) we plot which variables appear in the 1-UIP clause learnt after analyzing the conflict. We observe that, in general, conflicts relate variables of few communities. In fact, this clause mainly contains variables of the community where the solver is focusing its last decisions, and (very) few variables of the rest. Notice that all of these variables

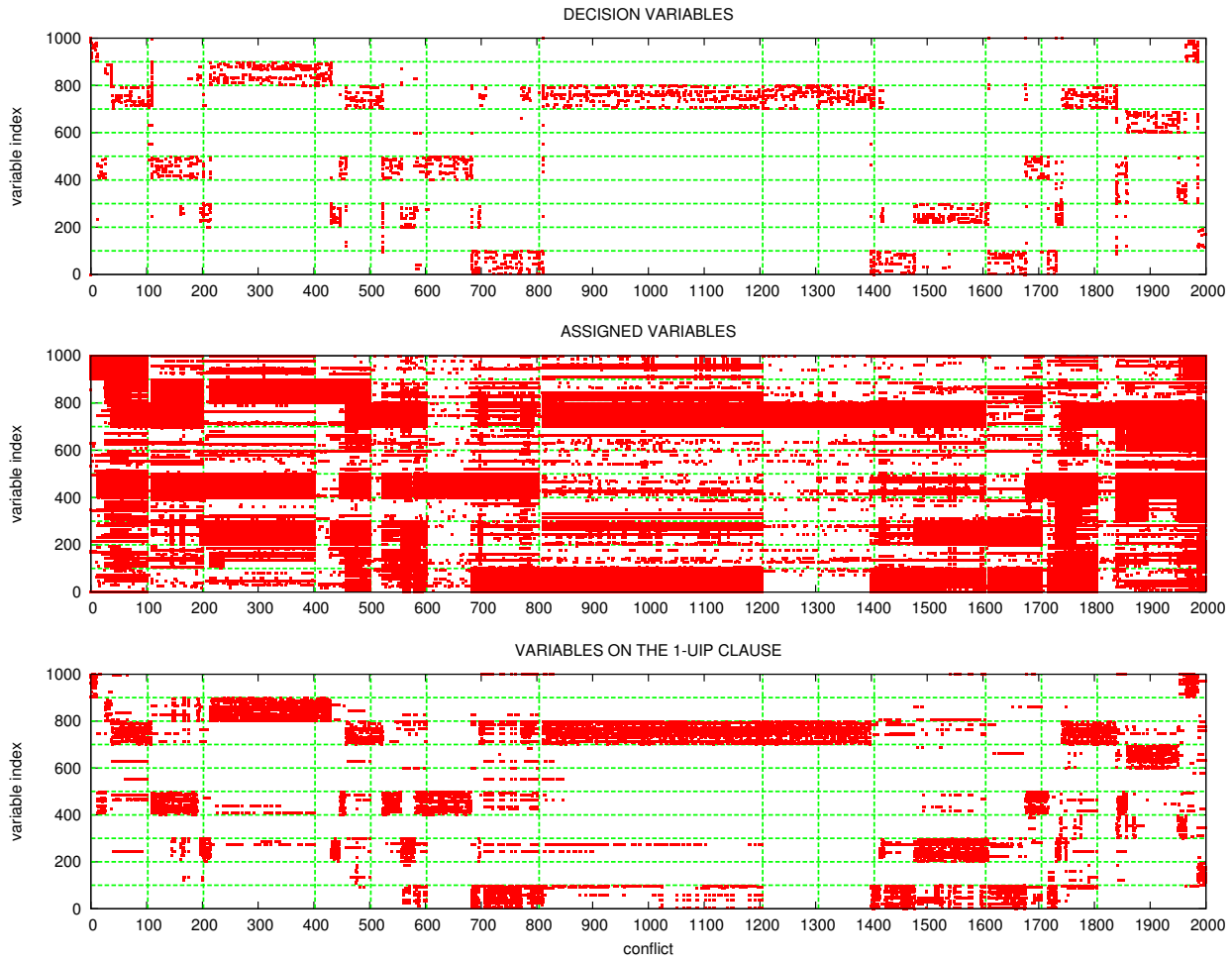

Figure 7: Decision variables (top), assigned variables (center) and variables on the learnt clause (bottom) along the first 2000 conflicts of the execution of MiniSAT 2.2 (i.e., with 1-UIP learning scheme) on a random instances with $n = 1000$, $m = 4200$, $k = 3$, $c = 10$ and $Q = 0.8$. Horizontal lines split the set of variables belonging to each community, and vertical lines represent the restarts.

had to be previously assigned. We also observe that in the last steps of the search (not shown in the plot), there exist many conflicts relating almost all communities.

In a second experiment, we want to investigate the relation between the clause learning techniques used by the SAT solver and the community structure of the formula. To this purpose, we modify the solver MiniSAT with another learning strategy: the *decision-induced clause learning scheme*. This strategy learns the decision variables that implies the conflict (i.e., it explores the whole implication graph of the conflict till the decision nodes). This is one of the most classical learning strategies, and it was proposed in GRASP [38].

In Figure 8, we solve the same instance of the previous experiment using a modified version of MiniSAT with a *decision-induced clause learning scheme*. As for the 1-UIP schema, we represent the decision variables (top), the assigned variables (center), and the variables belonging to the learnt clause (bottom), for the first 2000 conflicts. Using this learning strategy, the solver require 157726 decisions and 113012 conflicts to solve the instance, spending a total of 23.1 seconds. Notice that this is approximately one order of magnitude slower. The reason of this experiment is to show how CDCL techniques, when used all together, help the solver to focus on particular communities along the search. On the contrary, small changes in these techniques may provoke that community structure is not explicitly considered any more, affecting thereby the overall performance of the solver. This may explain the success of these

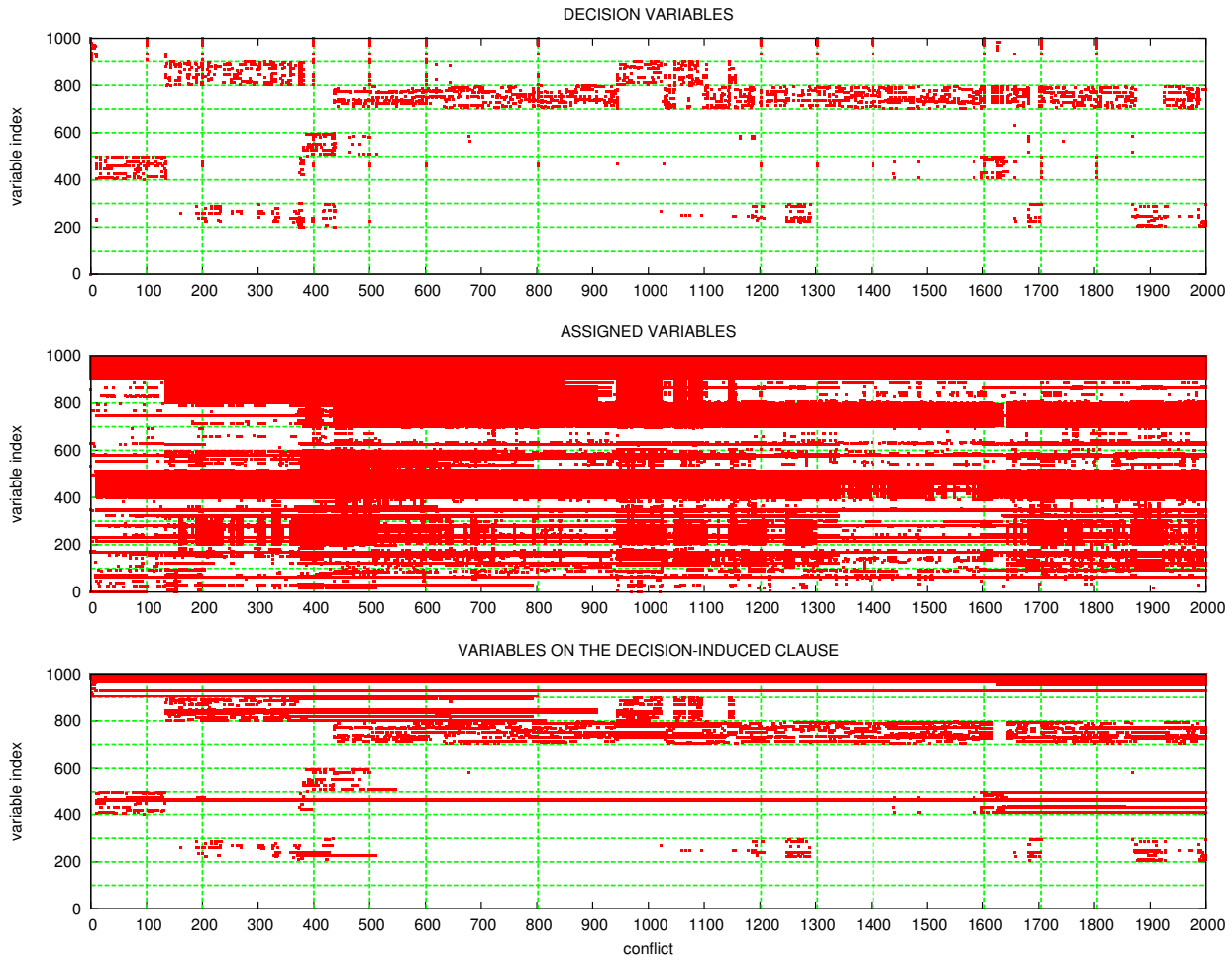

Figure 8: Decision variables (top), assigned variables (center) and variables on the learnt clause (bottom) along the first 2000 conflicts of the execution of a modified version of MiniSAT with a *decision-induced clause learning scheme*, on a random instances with $n = 1000$, $m = 4200$, $k = 3$, $c = 10$ and $Q = 0.8$. Horizontal lines split the set of variables belonging to each community, and vertical lines represent the restarts.

technique on benchmarks with a clear community structure, as industrial SAT instances are.

First, we observe that the solver also tends to focus its decisions on communities with this learning strategy. However, this phenomenon is less clear than in the previous experiment. For instance, between conflicts 200 and 400, the solver is focusing on communities c_3 and c_9 at the same time. This effect can be explained with the next observation. Second, we show that many restarts have no effect on most of the assigned variables. That means that even when restarts remove the value of all assigned variables, they are re-assigned again afterwards. See, for instance, community c_{10}: it is assigned at the beginning of the search, and it remains assigned during most of the execution. Finally, we observe that the variables belonging to the learnt clause correspond in fact to the variables of the communities where the solver was focused at the beginning of the search. Therefore, the activity of these variables are constantly increased, and hence, they are assigned once they become unassigned (i.e., after a restart). In fact, the presence of these variable in the learnt clause explains why they are re-assigned after each restart. These observations allow us to understand how the 1-UIP, in joint with the activity-based heuristics, help the solver to focus on communities, and this may explain the different performance (e.g., number of decisions and conflicts, runtime) spent to solve an industrial instance. All these effects cannot be observed if we use a formula with a low modularity because the communities are not so well delimited.

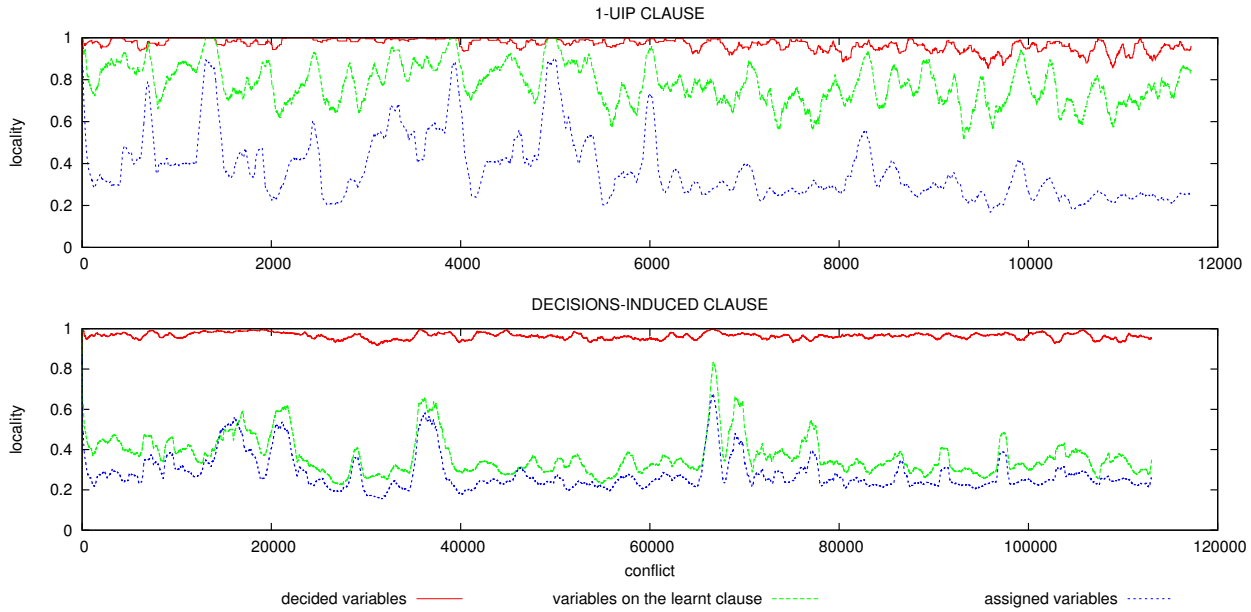

Figure 9: Evolution of the locality of the set of decided variables, set of assigned variables and learnt clause along the execution of 1-UIP based solver (top) and a decision-induced learning based solver (bottom).

In the last experiment, we want to analyze the *locality* of the solver during the search. In other words, we want to *measure* how much the solver focuses its decisions and learnt clauses on variables of the same community. In order to quantify this locality, we introduce the following definition, inspired by the notion of modularity.

Definition 4. *Given a subset of variables $S \subseteq N$, and a partition P of the variables $N = \cup_{i=1}^{c} P_i$, we define*

$$\text{locality}(S, P) = \frac{\sum_{i=1}^{c} |P_i \cap S|^2}{|S|^2}$$

Intuitively, this definition expresses the fraction of edges between nodes of the same community with respect to the total number of edges, if we consider all possible edges between vertexes of S (including self-loops). Notice that, if all variables of S are split into r communities of same size (i.e. $|P_i \cap S| = |S|/r$, for $i = 1, \ldots, r$, and $|P_i \cap S| = 0$, for $i = r+1, \ldots, c$), then $\text{locality}(S, P) = 1/r$. Therefore, the inverse of locality measures the number of distinct communities involved in a set of variables (when all of these communities contain the same number of variables of S).

In Figure 9, we represent the evolution of the locality of the set of decided variables (between two conflicts), the set of assigned variables (between two conflicts), and the set of variables on the learnt clause, along the execution of the solver for the same SAT instance as in previous experiments, using as learnt clause the 1-UIP and the decisions-induced clauses. We only represent the moving average every 100 conflicts for 1-UIP and 1000 conflicts for decisions-induced clauses. We observe that in both cases, the set of decided variables is very local, close to 1. This means that all variables decided between two conflicts almost always belong to the same community. This is not the case when we analyze the set of assigned variables. In this case, the average locality over the whole execution is 0.373 for the 1-UIP strategy and 0.281 for the other learning scheme. The locality of variables on the learnt clauses has a value in between the decision and the assigned variables. However, we clearly notice that, for the 1-UIP scheme this locality (0.782 on average) is much bigger than for decision-induced clauses (0.367 on average). We also notice that when the locality of the set of assigned variables increases (after a restart, for instance), the locality of variables on the learnt clauses also increases. Therefore, it is good for the solvers performance to increase the locality of assigned variables.

Finally, Norbert Manthey [28] reports good results on the use of our generator to train a configurable SAT solver to improve its performance. He used the pseudo-industrial random SAT instances created by our model and used in

the last SAT Race 2015 [17]. This family of instances contains 44 SAT formulas generated with $n = 2200$ variables, $m = 9086$ clauses, $c = 40$ communities and modularity $Q = 0.8$. He used this set of instances to train the SAT solver Riss 5.1.0 [23] (using the configuration tool SMAC [21]). Riss is a very configurable SAT solver. Then, this configuration was used in Riss to solve the aggregated set of industrial instances used in all SAT Competitions from 2002 to 2015, and its performance was compared to the performance of the default configuration of this solver on the same set. Interestingly, the resulting configuration (i.e., from training this solver with our pseudo-industrial SAT instances) performed better than its default configuration. This suggests that our model captures a very important feature of industrial SAT problems which is, in fact, crucial in the solving process.

9. Conclusions and Future Work

In the SAT community, it is accepted that industrial problems exhibit some kind of *structure* that is exploited by CDCL SAT solvers. Nowadays, one of the most intriguing questions is how to characterize this structure, with the aim of develop random SAT instances generation models that capture realistically the features of industrial problems, for SAT solving testing/analysis purposes. Recently, the notions of community structure and modularity have been used with success to explain the structure of SAT instances [7], and their hardness [34].

We present a modularity-based generator, which generates random k-CNF SAT instances of any desired modularity. Industrial problems are characterized by a high modularity. Therefore, our model can generate more realistic pseudo-industrial random formulas on demand. We validate the adequacy of this model checking that (i) the community structure of the resulting formulas is the expected, (ii) if there exists a phase transition point dependent on the clause/variable ratio, then it is independent on the modularity, and (iii) the SAT solvers performances are consistent to the structure of the formulas generated by our model, i.e. SAT solvers *specialized* in industrial (random) problems perform better in *high modular* (*low modular*) instances.

Finally, we use our generator to study how the community structure is affected by some components of the solver. Namely, we study the variables branching heuristics and the clause learning mechanism. We observe that, for a given period of time, the solver tends to focus its decisions on variables of the same community, and learns clauses mostly relating variables of this community. We also show that restarts help to unassign variables belonging to communities where the solver is no longer focused on. Therefore, the community structure of the instance plays an important role in order to explain the success of these techniques, when they are used all together. On the contrary, we see that the solver has a worse performance when it uses instead a learning strategy that does not take into account such structure, as the learning of the *decision-induced clause*.

The Community Attachment model forces some features of the resulting SAT instance to be as much regular as possible. In particular, all clauses have exactly the same number of literals (i.e., k literals), all communities approximately have the same number of variables (i.e., $\lfloor n/c \rfloor$ or $\lfloor n/c \rfloor + 1$), and all variables approximately have the same number of occurrences. This allows us to study the real impact of certain SAT solving techniques on the community structure without any undesired secondary effect. On the other hand, real application benchmarks are characterized by a certain variability in the clause size, community size, and number of variable occurrences. Therefore, some natural extensions of the Community Attachment model may consider these cases. A possibility would be to assign a distinct probability to each variable, as it is described in [6]. This would result into random instances with scale-free structure and high modularity, as observed in real-world instances.

References

[1] Achlioptas, D., Gomes, C., Kautz, H., Selman, B., 2000. Generating satisfiable problem instances. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00). pp. 256–261.

[2] Aldecoa, R., Orsini, C., Krioukov, D. V., 2015. Hyperbolic graph generator. Computer Physics Communications 196, 492–496.

[3] Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., Levy, J., 2014. The fractal dimension of SAT formulas. In: Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14). pp. 107–121.

[4] Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., Levy, J., 2015. On the classification of industrial SAT families. In: Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence (CCIA'15). pp. 163–172.

[5] Ansótegui, C., Bonet, M. L., Levy, J., 2009. On the structure of industrial SAT instances. In: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP'09). pp. 127–141.

[6] Ansótegui, C., Bonet, M. L., Levy, J., 2009. Towards industrial-like random SAT instances. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09). pp. 387–392.

[7] Ansótegui, C., Giráldez-Cru, J., Levy, J., 2012. The community structure of SAT formulas. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12). pp. 410–423.
[8] Ansótegui, C., Giráldez-Cru, J., Levy, J., Simon, L., 2015. Using community structure to detect relevant learnt clauses. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15). pp. 238–254.
[9] Audemard, G., Simon, L., 2009. Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09). pp. 399–404.
[10] Barabási, A. L., Albert, R., 1999. Emergence of scaling in random networks. Science 286, 509–512.
[11] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D., 2008. On modularity clustering. IEEE Trans. on Knowledge and Data Engineering 20 (2), 172–188.
[12] Burg, S., Kaufmann, M., Kottler, S., 2012. Creating industrial-like SAT instances by clustering and reconstruction. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12). pp. 471–472.
[13] Cook, S. A., 1971. The complexity of theorem-proving procedures. In: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71). pp. 151–158.
[14] Dechter, R., 2003. Constraint Processing. Morgan Kaufmann.
[15] Eén, N., Sörensson, N., 2003. An extensible SAT-solver. In: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03). pp. 502–518.
[16] Gent, I. P., Hoos, H. H., Prosser, P., Walsh, T., 1999. Morphing: Combining structure and randomness. In: Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99). pp. 654–660.
[17] Giráldez-Cru, J., Levy, J., 2015. Benchmarks description. In: Proceedings of the SAT Race 2015.
[18] Giráldez-Cru, J., Levy, J., 2015. A modularity-based random SAT instances generator. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 1952–1958.
[19] Gomes, C. P., Selman, B., 1997. Problem structure in the presence of perturbations. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97). pp. 221–226.
[20] Heule, M. J. H., van Zwieten, J. E., Dufour, M., van Maaren, H., 2004. March_eq: Implementing additional reasoning into an efficient Look-ahead SAT solver. In: Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04). pp. 345–359.
[21] Hutter, F., Hoos, H. H., Leyton-Brown, K., 2011. Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th Conference on Learning and Intelligent Optimization (LION-5). pp. 507–523.
[22] Järvisalo, M., Kaski, P., Koivisto, M., Korhonen, J. H., 2012. Finding efficient circuits for ensemble computation. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12). pp. 369–382.
[23] Kahlert, L., Krüger, F., Manthey, N., Stephan, A., 2015. Riss solver framework v5.05. In: Proceedings of the SAT Race 2015.
[24] Katsirelos, G., Simon, L., 2012. Eigenvector centrality in industrial SAT instances. In: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12). pp. 348–356.
[25] Kautz, H. A., Selman, B., 2003. Ten challenges redux: Recent progress in propositional reasoning and search. In: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03). pp. 1–18.
[26] Liang, J. H., Ganesh, V., Poupart, P., Czarnecki, K., 2015. Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In: Proceedings of the 30th National Conference on Artificial Intelligence (AAAI'16). pp. 225–241.
[27] Liang, J. H., Ganesh, V., Zulkoski, E., Zaman, A., Czarnecki, K., 2015. Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In: Proceedings of the 11th International Haifa Verification Conference on Hardware and Software: Verification and Testing (HVC'15). pp. 225–241.
[28] Manthey, N., 2015. Personal communication.
[29] Martins, R., Manquinho, V. M., Lynce, I., 2013. Community-based partitioning for MaxSAT solving. In: Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT'13). pp. 182–191.
[30] Mitchell, D., Selman, B., Levesque, H., 1992. Hard and easy distributions of SAT problems. In: Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92). pp. 459–465.
[31] Neves, M., Martins, R., Janota, M., Lynce, I., Manquinho, V. M., 2015. Exploiting resolution-based representations for maxSAT solving. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15). pp. 272–286.
[32] Newman, M. E. J., 2004. Fast algorithm for detecting community structure in networks. Phys. Rev. E 69 (6), 066133.
[33] Newman, M. E. J., Girvan, M., 2004. Finding and evaluating community structure in networks. Phys. Rev. E 69 (2), 026113.
[34] Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., Simon, L., 2014. Impact of community structure on SAT solver performance. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'14). pp. 252–268.
[35] Newsham, Z., Lindsay, W., Ganesh, V., Liang, J. H., Fischmeister, S., Czarnecki, K., 2015. SATGraf: Visualizing the evolution of SAT formula structure in solvers. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15). pp. 62–70.
[36] Papadopoulos, F., Kitsak, M., Serrano, M., Boguñá, M., Krioukov, D., Sep 2012. Popularity versus Similarity in Growing Networks. Nature 489, 537–540.
[37] Selman, B., Kautz, H. A., McAllester, D. A., 1997. Ten challenges in propositional reasoning and search. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). pp. 50–54.
[38] Silva, J. P. M., Sakallah, K. A., 1996. GRASP - a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'96). pp. 220–227.
[39] Slater, A., 2002. Modelling more realistic SAT problems. In: Proceedings of the 15th Australian Joint Conference on Artificial Intelligence (AJCAI'02). pp. 591–602.
[40] Sonobe, T., Kondoh, S., Inaba, M., 2014. Community branching for parallel portfolio SAT solvers. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'14). pp. 188–196.
[41] Walsh, T., 1999. Search in a small world. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99). pp.

1172–1177.