The Community Structure of SAT Formulas*

Carlos Ansótegui[1], Jesús Giráldez-Cru[2], and Jordi Levy[2]

[1] DIEI, Univ. de Lleida
carlos@diei.udl.cat
[2] IIIA-CSIC
{jgiraldez,levy}@iiia.csic.es

Abstract. The research community on complex networks has developed techniques of analysis and algorithms that can be used by the SAT community to improve our knowledge about the structure of industrial SAT instances. It is often argued that modern SAT solvers are able to exploit this *hidden structure*, without a precise definition of this notion.
In this paper, we show that most industrial SAT instances have a high modularity that is not present in random instances. We also show that successful techniques, like learning, (indirectly) take into account this community structure. Our experimental study reveal that most learnt clauses are local on one of those modules or communities.

1 Introduction

In recent years, SAT solvers efficiency solving industrial instances has undergone a great advance, mainly motivated by the introduction of lazy data-structures, learning mechanisms and activity-based heuristics [11, 18]. This improvement is not shown when dealing with randomly generated SAT instances. The reason for this difference seems to be the existence of a structure in industrial instances [25].

In parallel, there have been significant advances in our understanding of complex networks, a subject that has focused the attention of statistical physicists. The introduction of these network analysis techniques could help us to understand the nature of SAT instances, and could contribute to further improve the efficiency of SAT solvers. Watts and Strogatz [24] introduce the notion of *small world*, the first model of complex networks, as an alternative to the classical random graph models. Walsh [23] analyzes the small world topology of many graphs associated with search problems in AI. He also shows that the cost of solving these search problems can have a *heavy-tailed distribution*. Gomes et al. [14, 15] propose the use of *randomization* and *rapid restart* techniques to prevent solvers from falling on the long tail of such kinds of distributions.

The notion of structure has been addressed in previous work [14, 16, 13, 17, 3]. In [22] it is proposed a method to generate more realistic random SAT problems based on the notions of *characteristic path length* and *clustering coefficient*. Here

* This research has been partially founded by the CICYT research projects TASSAT (TIN2010-20967-C04-01/03) and ARINF (TIN2009-14704-C03-01).

we use a distinct notion of modularity. In [6], it is shown that many SAT instances can be decomposed into *connected components*, and how to handle them within a SAT solver. They discuss how this component structure can be used to improve the performance of SAT solvers. However, their experimental investigation shows that this is not enough to solve more efficiently SAT instances. The notion of *community* is more general than the notion of *connected components*. In particular, it allows the existence of (a few) connections between communities. As we discus later, industrial SAT instances use to have a connected component containing more than the 99% of the variables. Also, in [1] some techniques are proposed to reason with multiple knowledge bases that overlap in content. In particular, they discuss strategies to induce a partitioning of the axioms, that will help to improve the efficiency of reasoning.

In this paper we propose the use of techniques for detecting the *community structure* of SAT instances. In particular, we apply the notion of *modularity* [19] to detect these communities. We also discuss how existing conflict directed clause learning algorithms and activity-based heuristics already take advantage, *indirectly*, of this community structure. Activity-based heuristics [18] rely on the idea of giving higher priority to the variables that are involved in (recent) conflicts. By focusing on a sub-space, the covered spaces tend to coalesce, and there are more opportunities for resolution since most of the variables are common.

2 Preliminaries

Given a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, a *literal* is an expression of the form x_i or $\neg x_i$. A *clause* c of length s is a disjunction of s literals, $l_1 \vee \ldots \vee l_s$. We say that s is the size of c, noted $|c|$, and that $x \in c$, if c contains the literal x or $\neg x$. A *CNF formula* or *SAT instance* of length t is a conjunction of t clauses, $c_1 \wedge \ldots \wedge c_t$.

An (undirected) graph is a pair (V, w) where V is a set of vertices and $w : V \times V \to \mathbb{R}^+$ satisfies $w(x, y) = w(y, x)$. This definition generalizes the classical notion of graph (V, E), where $E \subseteq V \times V$, by taking $w(x, y) = 1$ if $(x, y) \in E$ and $w(x, y) = 0$ otherwise. The degree of a vertex x is defined as $\deg(x) = \sum_{y \in V} w(x, y)$. A bipartite graph is a tuple (V_1, V_2, w) where $w : V_1 \times V_2 \to \mathbb{R}^+$.

Given a SAT instance, we construct two graphs, following two models. In the Variable Incidence Graph model (VIG, for short), vertices represent variables, and edges represent the existence of a clause relating two variables. A clause $x_1 \vee \ldots \vee x_n$ results into $\binom{n}{2}$ edges, one for every pair of variables. Notice also that there can be more than one clause relating two given variables. To preserve this information we put a higher weight on edges connecting variables related by more clauses. Moreover, to give the same relevance to all clauses, we ponderate the contribution of a clause to an edge by $1/\binom{n}{2}$. This way, the sum of the weights of the edges generated by a clause is always one. In the Clause-Variable Incidence Graph model (CVIG, for short), vertices represent either variables or clauses, and edges represent the occurrence of a variable in a clause. Like in the VIG model, we try to assign the same relevance to all clauses, thus every edge

connecting a variable x with a clause C containing it has weight $1/|C|$. This way, the sum of the weights of the edges generated by a clause is also one in this model.

Definition 1 (Variable Incidence Graph (VIG)). *Given a SAT instance Γ over the set of variables X, its variable incidence graph is a graph (X, w) with set of vertices the set of Boolean variables, and weight function:*

$$w(x,y) = \sum_{\substack{c\in\Gamma \\ x,y\in c}} \frac{1}{\binom{|c|}{2}}$$

Definition 2 (Clause-Variable Incidence Graph (CVIG)). *Given a SAT instance Γ over the set of variables X, its clause-variable incidence graph is a bipartite graph $(X, \{c \mid c \in \Gamma\}, w)$, with vertices the set of variables and the set of clauses, and weight function:*

$$w(x,c) = \begin{cases} 1/|C| & \textit{if } x \in c \\ 0 & \textit{otherwise} \end{cases}$$

3 Modularity in Large-Scale Graphs

To analyze the structure of a SAT instance we will use the notion of *modularity* introduced by [20]. This property is defined for a graph and a specific *partition* of its vertices into *communities*, and measures the adequacy of the partition in the sense that most of the edges are within a community and few of them connect vertices of distinct communities. The modularity of a graph is then the maximal modularity for all possible partitions of its vertices. Obviously, measured this way, the maximal modularity would be obtained putting all vertices in the same community. To avoid this problem, Newman and Girvan define modularity as the fraction of edges connecting vertices of the same community minus the expected fraction of edges for a random graph with the same number of vertices and same degree.

Definition 3 (Modularity of a Graph). *Given a graph $G = (V, w)$ and a partition $P = \{P_1, \ldots, P_n\}$ of its vertices, we define their* modularity *as*

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

We call the first term of this formula the inner edges fraction*, IEF for short, and the second term the* expected inner edges fraction*, IEF^e for short. Then, $Q = IEF - IEF^e$.*

The (optimal) modularity *of a graph is the maximal modularity, for any possible partition of its vertices: $Q(G) = \min\{Q(G,P) \mid P\}$*

Since the IEF and the IEF^e of a graph are both in the range $[0, 1]$, and, for the partition given by a single community, both have value 1, the optimal modularity of graph will be in the range $[0, 1]$. In practice, Q values for networks showing a strong community structure range from 0.3 to 0.7, higher values are rare [20].

There has not been an agreement on the definition of modularity for bipartite graphs. Here we will use the notion proposed by [4] that extends Newman and Girvan's definition by restricting the random graphs used in the computation of the IEF^{e}to be bipartite. In this definition, communities may contain vertices of V_1 and of V_2.

Definition 4 (Modularity of a Bipartite Graph). *Given a graph* $G = (V_1, V_2, w)$ *and a partition* $P = \{P_1, \ldots, P_n\}$ *of its vertices, we define their* modularity *as*

$$Q(G,P) = \sum_{P_i \in P} \frac{\sum\limits_{\substack{x \in P_i \cap V_1 \\ y \in P_i \cap V_2}} w(x,y)}{\sum\limits_{\substack{x \in V_1 \\ y \in V_2}} w(x,y)} - \frac{\sum\limits_{x \in P_i \cap V_1} deg(x)}{\sum\limits_{x \in V_1} deg(x)} \cdot \frac{\sum\limits_{y \in P_i \cap V_2} deg(y)}{\sum\limits_{y \in V_2} deg(y)}$$

There exist a wide variety of algorithms for computing the modularity of a graph. Moreover, there exist alternative notions and definitions of modularity for analyzing the community structure of a network. See [12] for a survey in the field. The decision version of modularity maximization is NP-complete [8]. All the modularity-based algorithms proposed in the literature return an approximated lower bound for the modularity. They include greedy methods, methods based on simulated annealing, on spectral analysis of graphs, etc. Most of them have a complexity that make them inadequate to study the structure of an industrial SAT instance. There are algorithms specially designed to deal with large-scale networks, like the greedy algorithms for modularity optimization [19, 9], the label propagation-based algorithm [21] and the method based on graph folding [7].

The first described algorithm for modularity maximization is a *greedy method* of Newman [19]. This algorithm starts by assigning every vertex to a distinct community. Then, it proceeds by joining the pair of communities that result in a bigger increase of the modularity value. The algorithm finishes when no community joining results in an increase of the modularity. In other words, it is a greedy gradient-guided optimization algorithm. The algorithm may also return a dendogram of the successive partitions found. Obviously, the obtained partition may be a local maximum. In [9] the data structures used in this basic algorithm are optimized, using among other data structures for sparse matrices. The complexity of this refined algorithm is $\mathcal{O}(m\,d\,\log n)$, where d is the depth of the dendogram (i.e. the number of joining steps), m the number of edges and n the number of vertices. They argue that d may be approximated by $\log n$, assuming that the dendogram is a balanced tree, and the sizes of the communities are similar. However, this is not true for the graphs we have analyzed, where the

sizes of the communities are not homogeneous. This algorithm has not been able to finish, for none of our SAT instances, with a run-time limit of one hour.

An alternative algorithm is the *Label Propagation Algorithm (LPA)* proposed by [21] (see Algorithm 1). Initially, all vertices are assigned to a distinct label, e.g., its identifier. Then, the algorithm proceeds by re-assigning to every vertex the label that is more frequent among its neighbors. The procedure ends when every vertex is assigned a label that is maximal among its neighbors. The order in which the vertices update their labels in every iteration is chosen randomly. In case of a tie between maximal labels, the winning label is also chosen randomly. The algorithm returns the partition defined by the vertices sharing the same label. The label propagation algorithm has a near linear complexity. However, it has been shown experimentally that the partitions it computes have a worse modularity than the partitions computed by the Newman's greedy algorithm.

The *Graph Folding Algorithm (GFA)* proposed in [7] (see Algorithm 2) improves the Label Propagation Algorithm in two directions. The idea of moving one node from one community to another following a greedy strategy is the same, but, instead of selecting the community where the node has more neighbors, it selects the community where the movement would most increase the modularity. Second, once no movement of node from community to community can increase the modularity (we have reached a (local) modularity maximum), we allow to merge communities. For this purpose we construct a new graph where nodes are the communities of the old graph, and where edges are weighted with the sum of the weights of the edges connecting both communities. Then, we apply again the greedy algorithm to the new graph. This folding process is repeated till no modularity increase is possible.

4 Modularity of SAT Instances

We have computed the modularity of the SAT instances used in the 2010 SAT Race Finals (see `http://baldur.iti.uka.de/sat-race-2010/`). They are 100 instances grouped into 16 families. These families are also classified as cryptography, hardware verification, software verification and mixed, according to their application area. All instances are *industrial*, in the sense that their solubility has an industrial or practical application. However, they are expected to show a distinct nature.

We have observed that all instances of the same family have a similar modularity. Therefore, in Table 1, we only show the median of these values. We present the modularities obtained by LPA on the graphs VIG and CVIG, and by GFA on the graphs VIG. We have re-implemented both algorithms, and in the case of LPA, we have developed a new algorithm adapted for bi-partite graphs. The GFA algorithm is not yet adapted for bipartite graphs. We also study the *connected components* as in [6].

We have to remark that both algorithms give a lower bound on the modularity, hence we can take the maximum of both measures as a lower bound. Having this in mind, we can conclude that, except for the grieu family, all families show

Algorithm 1: Label Propagation Algorithm (LPA). The function most_freq_label returns the label that is most frequent among a set of vertices. In case of tie, it randomly chooses one of the maximal labels.

```
1  function most_freq_label(v, N)
2      SL :={L[v] | v ∈ N};
3      for l ∈ SL do
4          freq[l] := Σ_{v'∈N, l=L[v']} w(v, v')
5      Max := {l ∈ SL | freq[l] = max{freq[l] | l ∈ SL}};
6      return random_choose(Max)
7
   Input: Graph G = (X, w)
   Output: Label L
8  for x ∈ X do L[x] := x; freq[x] := 0;;
9  repeat
10     ord := shuffle(X);
11     changes := false;
12     for i ∈ X do
13         (l, f) := most_freq_label(i,neighbors(i));
14         changes := changes ∨ f > freq[i];
15         L[i] := l;
16         freq[i] := f
17 until ¬changes ;
```

a clear community structure with values of Q around 0.8. In other kind of networks values greater than 0.7 are rare, therefore the values obtained for SAT instances can be considered as exceptionally high.

As one could expect, we obtain better values with GFA than with the LPA algorithm. The reason for this better performance is that, whereas in the LPA we use the most frequent label among neighbors (in order to assign a new community to a node), in the GFA we select the label leading to a bigger increase in the modularity. The latter is clearly a better strategy for obtaining a bigger resulting modularity. Moreover, in the GFA a further step is added where communities can be merged, when no movement of a single node from one community to another leads to a modularity increase.

If we compare the modularity values for the VIG model (obtained with the LPA) with the same values for the CVIG model, we can conclude that, in general, these values are higher for the CVIG model. It could be concluded that the loss of information, during the *projection* of the bipartite CVIG graph into the VIG graph, may destroy part of the modular structure. However, this is not completely true. Suppose that the instance has no modular structure at all, but all clauses are binary. We can construct a partition as follows: put every variable into a distinct community, and every clause into the same community of one of its variables. Using this partition, half of the edges will be internal, i.e. IEF = 0.5, IEF^e will be nearly zero, and $Q \approx 0.5$. Therefore, we have to take into account that using Barber's modularity definition for bipartite graphs, as

Algorithm 2: Graph Folding algorithm (GFA)

```
1  function OneLevel(GraphG = (X, w)) : Label L
2      foreach i ∈ X do L[i] := i  repeat
3          changes := false;
4          foreach i ∈ X do
5              bestinc := 0;
6              foreach c ∈ {c | ∃j.w(i, j) ≠ 0 ∧ L[j] = c} do
7                  inc :=
                   Σ_{L(j)=c} w(i, j) − arity(i) · Σ_{L[j]=c} arity(j)/ Σ_{j∈X} arity(j);
8                  if inc > bestinc then
9                      L[i] := c; bestinc := inc; changes := true;
10     until ¬changes ;
11     return L;
12
13 function Fold(Graph G_1, Label L) : Graph G_2
14     X_2 = {c | ∀i, j ∈ c.L[i] = L[j];
15     w_2(c_1, c_2) = Σ_{i∈c_1, j∈c_2} w_2(i, j);
16     return G_2 = (X_2, w_2);
17
   Input: Graph G = (X, w)
   Output: Label L_1
18 foreach i ∈ X do L_1[i] := i;
19 L_2 := OneLevel(G);
20 while Modularity(G, L_1) < Modularity(G, L_2) do
21     L_1 := L1 ∘ L2;
22     G = Fold(G, L_2);
23     L_2 := OneLevel(G);
```

we do, if vertex degrees are small, modularity can be quite big compared with Newman's modularity.

We also report results on the number of communities ($|P|$) and the fraction of vertices belonging to the largest community ($larg$) expressed as a percentage. If all communities have a similar size, then $larg \approx 1/|P|$. In some cases, like palacios and mizh, we have $|P| \gg 1/larg$. This means that the community structure corresponds to a big (or few) big central communities surrounded by a multitude of small communities. In some cases, the sizes of communities seem to follow a power-law distribution (this is something we would have to check). The existence of a big community implies an expected inner fraction close to one, hence a modularity close to zero.

In both algorithms, in every iteration we have to visit all neighbors of every node. Therefore, the cost of an iteration is linear in the number of edges of the graph. We observe although than GFA usually needs more iterations than LPA. This is because, after folding the graph, we can do further iterations, and even several graph foldings.

Family (#instanc.)		Variable IG								Clause-Variable IG				Connect. Comp.	
		LPA				GFA				LPA					
		Q	$\|P\|$	larg.	iter.	Q	$\|P\|$	larg.	iter.	Q	$\|P\|$	larg.	iter.	$\|P\|$	larg.
cripto.	desgen(4)	**0.88**	532	0.8	36	**0.95**	97	2	37	**0.75**	3639	1	25	1	100
	md5gen(3)	**0.61**	7176	0.1	16	**0.88**	38	7	40	**0.78**	7904	0.1	42	1	100
	mizh(8)	0.00	33	99	6	**0.74**	30	9	33	**0.67**	5189	31	43	1	100
hard. ver.	ibm(4)	**0.81**	3017	0.6	9	**0.95**	723	4	32	**0.77**	19743	0.2	140	70	99
	manolios(16)	0.30	66	81	9	**0.89**	37	9	81	**0.76**	6080	1	26	1	100
	velev (10)	**0.47**	8	68	9	**0.69**	12	30	24	0.30	1476	77	31	1	100
mixed	anbulagan(8)	**0.55**	30902	0.1	11	**0.91**	90	2	43	**0.72**	46689	0.6	26	1	100
	bioinf(6)	**0.61**	87	44	3	**0.67**	60	17	22	**0.64**	94027	15	10	1	100
	diagnosis(4)	**0.61**	20279	0.7	15	**0.95**	68	3	43	**0.65**	85928	0.1	42	1	100
	grieu(3)	0	1	100	2	0.23	9	14	11	0	1	100	14	1	100
	jarvisalo(1)	**0.57**	260	5	8	**0.76**	19	9	26	**0.71**	336	1	11	1	100
	palacios(3)	0.14	1864	96	58	**0.93**	1802	6	13	**0.76**	2899	0.4	35	1	100
soft. ver.	babic(2)	**0.68**	34033	8	54	**0.90**	6944	10	141	**0.73**	59743	4	53	41	99
	bitverif(5)	**0.48**	3	57	4	**0.87**	24	6	29	**0.76**	33276	0.4	8	1	100
	fuhs(4)	0.02	18	99	43	**0.81**	43	7	30	**0.67**	12617	0.8	28	1	100
	nec(17)	0.07	107	96	31	**0.93**	65	14	124	**0.79**	23826	0.8	114	1	100
	post(2)	**0.36**	$3{\cdot}10^6$	53	54	**0.81**	$3{\cdot}10^6$	9	262	**0.72**	$3{\cdot}10^6$	6	49	224	99

Table 1. Modularity of 2010 SAT Race instances, using LPA and GFA. Q stands for modularity, $|P|$ for number of communities, *larg.* for fraction of vertices in the largest community, and *iter.* for number of iterations of the algorithm.

We have also studied the *connected components* of these instances as in [6]. As we can see, almost all instances have a single connected component, or almost all variables are included in the same one. Hence the rest of connected components contain just a few variables. Therefore, the modularity gives us much more information about the structure of the formula. Notice that a connected component can be structured into several communities.

In Table 2 we show the results for the modularity computed after preprocessing the formula with the Satelite preprocessor [10]. The modularities are computed using the GFA algorithm. Satelite is an algorithm that applies variable elimination techniques. We can see that these transformations almost does not affect to the modularity of the formula. However, it eliminates almost all the small unconnected components of the formula.

5 Modularity of the Learnt Clauses

Most modern SAT solvers, based on variants of the DPLL schema, transform the formula during the proof or the satisfying assignment search. Therefore, the natural question is: even if the original formula shows a community structure, could it be the case that this structure is quickly destroyed during the search process? Moreover, most SAT solvers also incorporate learning techniques that introduce new learnt clauses to the original formula. Therefore, a second question

Family		Orig. Form.	Preprocessed Formula				
			Modularity			Connect. Comp.	
		Q	Q	$\|P\|$	larg.	$\|P\|$	larg.
cripto.	desgen (4)	**0.951**	**0.929**	81	3.1	1	100
	md5gen (3)	**0.884**	**0.884**	18	8.5	1	100
	mizh (8)	**0.741**	**0.741**	18	9.5	1	100
hard. ver.	ibm (4)	**0.950**	**0.905**	26	6.1	1	100
	manolios (16)	**0.890**	**0.800**	16	14.9	1	100
	velev (10)	**0.689**	**0.687**	6	30.3	1	100
mixed	anbulagan (8)	**0.909**	**0.913**	47	5.1	1	100
	bioinf (6)	**0.673**	**0.657**	25	11.1	2	99.9
	diagnosis (4)	**0.952**	**0.950**	65	3.6	1	100
	grieu (3)	0.235	0.235	9	14.3	1	100
	jarvisalo (1)	**0.758**	**0.722**	11	13.1	1	100
	palacios (3)	**0.928**	**0.848**	17	10.76	1	100
soft. ver.	babic (2)	**0.901**	**0.875**	23	9.7	1	100
	bitverif (5)	**0.875**	**0.833**	19	7.3	1	100
	fuhs (4)	**0.805**	**0.743**	32	9.6	1	100
	nec (17)	**0.929**	**0.879**	37	10.3	1	100

Table 2. Modularity (computed with GFA) after and before preprocessing the formula with the Satelite preprocessor [10]

is: how these new clauses affect to the community structure of the formula? Finally, even if the value of the modularity is not altered, it can be the case that the communities are changed.

We have conducted a series of experiments to answer to the previous questions. We use the picosat SAT solver [5] (version 846), since it incorporates a conflict directed clause learning algorithm, activity-based heuristics, and restarting strategies.

In Table 3 we show the values of the original modularity compared with the modularity obtained after adding the learnt clauses to the original formula. We can observe that the modularity weakly decreases with the learnt clauses, but it is still meaningful. Therefore, learning does not completely destroy the organization of the formula into weakly connected communities.

The question now is, even if the modularity does not decreases very much, could it be the case that the communities have changed? In other words, there are still communities, but are they distinct communities?

If a considerable part of learning is performed locally inside one or a few communities, then the communities will not change. We have conducted another experiment to see if this is true. For the VIG model, we use the original formula to get a partition of the vertices, i.e. of the variables, into communities. Then, we use modularity as a *quality measure* to see how good is the same partition, applied to the graph obtained from the set of learnt clauses. Notice that modularity is a

Family		Orig. Form.	Orig. + Learnt Formula Modularity		
		Q	Q	$\|P\|$	larg.
cripto.	desgen (2)	**0.951**	**0.561**	53	13.0
	md5gen (3)	**0.884**	**0.838**	19	8.0
	mizh (1)	**0.741**	**0.705**	28	11.4
hard. ver.	ibm (3)	**0.950**	**0.912**	752	6.7
	manolios (14)	**0.890**	**0.776**	31	11.2
	velev (1)	**0.689**	**0.558**	6	30.1
mixed	anbulagan (4)	**0.909**	**0.876**	84	2.6
	bioinf (5)	**0.673**	**0.287**	32	58.7
	grieu (3)	0.235	0.085	6	35.0
	palacios (2)	**0.928**	**0.851**	2289	7.4
soft. ver.	babic (2)	**0.901**	**0.904**	6942	10.7
	fuhs (1)	**0.805**	**0.670**	24	7.6
	nec (15)	**0.929**	**0.936**	62	7.3

Table 3. Modularity (computed with GFA) of the original formula, and of the original formula with learnt clauses included.

	VIG			CVIG		
Family	orig.	first 100	all	orig	first 100	all
desgen (1)	**0.89**	**0.74**	0.08	**0.77**	0.28	0.09
md5gen (1)	**0.61**	**0.74**	0.02	**0.78**	**0.96**	0.02
ibm (2)	**0.84**	**0.60**	**0.47**	**0.81**	**0.58**	0.29
manolios (10)	0.21	0.04	0.10	**0.76**	0.11	0.09
anbulagan (2)	**0.56**	0.16	0.01	**0.87**	0.10	0.04
bioinf (4)	**0.62**	**0.46**	0.06	**0.68**	**0.69**	0.15
grieu (1)	0.00	0.00	0.00	0.00	0.00	0.00
babic (2)	**0.68**	**0.36**	**0.36**	**0.71**	**0.33**	**0.33**
fuhs (1)	**0.66**	**0.59**	0.14	**0.71**	**0.78**	0.07
nec (10)	0.12	0.01	0.12	**0.78**	**0.49**	0.24

Table 4. Modularity (computed by LPA) of the formula containing the first 100 learnt clauses, and all learnt clauses. In the first column we show the modularity of the original formula.

function of two parameters, in this case: the graph is the graph *containing* the learnt clauses, and the partition is computed for the formula *without* the learnt clauses. Since both graphs (the original formula and the learnt clauses) have the same set of vertices (the set of variables), this can be done directly.

For the CVIG model we must take into account that the graph contains variables and clauses as vertices. Therefore, the procedure is more complicated. We use the original formula to get a partition. We remove from this partition all clauses, leaving the variables. Then, we construct the CVIG graph for the set of learnt clauses. The partition classifies the variables of this second graph

Family	VIG			CVIG		
	orig.	first 100	all	orig.	first 100	all
md5gen (1)	**0.61**	**0.74**	0.02	**0.78**	**0.96**	0.02
ibm (2)	**0.84**	**0.50**	**0.43**	**0.81**	**0.58**	**0.42**
anbulagan (1)	**0.55**	0.03	0.00	**0.87**	0.13	0.05
manolios (9)	0.20	0.07	0.10	**0.76**	0.27	0.15
nec (10)	0.12	0.05	0.05	**0.78**	**0.30**	0.22

Table 5. Modularity (computed by LPA) of the learnt clauses that have been contributed to prove the unsatisfiability of the original formula. Like in Table 4 we show results for the first 100 learnt clauses, and for all clauses. We only show results for unsatisfiable formulas.

into communities, but not the clauses. To do this, we assign to each clause the community of variables where it has more of its variables included. In other words, given the labels of the variables we apply a single iteration of the label propagation algorithm to find the labels of the clauses.

We want to see how fast is the community structure degraded along the execution process of a SAT solver. Therefore, we have repeated the experiment for just the first 100 learnt clauses and for all the learnt clauses. We also want to know the influence of the quality of the learnt clauses. Therefore, we also repeat the experiment for all the learnt clauses (Table 4), and only using the clauses that participate in the proof of unsatisfiability (Table 5). Notice that Table 5 contains fewer entries than Table 4 because we can only consider unsatisfiable instances. Notice also that picosat is not able to solve all 2010 SAT Race instances, therefore Tables 4 and 5 contain fewer instances than Table 1. The analysis of the tables shows us that the CVIG model gives better results for the original formula and the first 100 learnt clauses, but equivalent results if we consider all learnt clauses. There are not significant differences if we use all learnt clauses, or just the clauses that participate in the refutation. Finally, there is a drop-off in the modularity (in the quality of the original partition) as we incorporate more learnt clauses. This means that, if we use explicitly the community structure to improve the efficiency of a SAT solver, to overcome this problem, we would have to recompute the partition (after some number of variable assignments or after a unit clause is learnt) to adjust it to the modified formula, like in [6].

It is worth to remark that, for the experiments in Table 4, the modularity for the VIG and the CVIG models, and the first 100 learnt clauses, is respectively 0.72 and 0.59. This means that, in the VIG model, around 72% of the first 100 learnt clauses could also be learnt working locally in each one of the communities. However, the percentage of learnt clauses that connect distinct communities is very significant.

6 Modularity of Random Formulas

We have also conducted a study of the modularity of 100 random 3-CNF SAT instances of 10^4 variables for different clause variable ratios (α). For this experi-

n	α	Q	$\|P\|$	larg.	iter
10^4	1	0.486	545	3.8	54
10^4	1.5	0.353	146	5.1	52
10^4	2	0.280	53	6.8	51
10^4	3	0.217	14	15.5	64
10^4	4	0.178	11	14.8	54
10^4	4.25	0.170	11	14.6	53
10^4	4.5	0.163	11	14.7	53
10^4	5	0.152	11	14.3	51
10^4	6	0.133	12	13.9	53
10^4	7	0.120	10	15.0	56
10^4	8	0.138	6	25.0	50
10^4	9	0.130	6	24.3	49
10^4	10	0.123	6	24.4	47

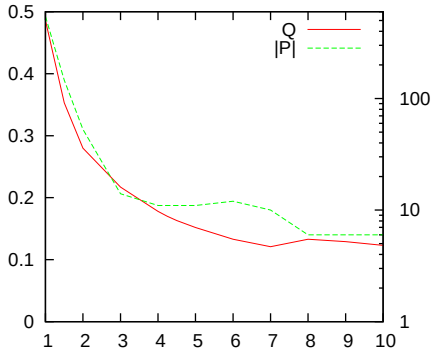

Table 6. Modularity (computed with GFA) of random formulas varying the clause variable ratio (α), and for $n = 10^4$ variables.

n	α	Q	$\|P\|$	larg.	iter
10^2	4.25	0.177	6	14.5	11
10^3	4.25	0.187	10.5	11.4	35
10^4	4.25	0.170	11	12.2	53
10^5	4.25	0.151	14	6.8	102
10^6	4.25	0.151	14	5.7	167

Table 7. Modularity (computed with GFA) of random formulas at the peak transition region (clause variable ratio α=4.25), varying the number of variables (n).

ment we used the GFA on the VIG model. Table 6 shows the results. As we can see, the modularity of random instances is only significant for very low variable ratios, i.e., on the leftist SAT easy side. This is due to the presence of a large quantity of very small communities. Notice, that as α increases, the variables get more connected but without following any particular structure, and the number of communities highly decreases. Even for low values of α, the modularity is not as high as for industrial instances, confirming their distinct nature. We do not observe any abrupt change in the phase transition point.

As a second experiment with random formulas, we wanted to investigate the modularity at the peak transition region for an increasing number of variables. Table 7 shows the results. As we can see, the modularity is very low and it tends to slightly decrease as the number of variables increases, and seems to tend to a particular value (0.15 for the phase transition point).

Finally, as with industrial instances, we wanted to evaluate the impact on modularity of the preprocessing with Satelite [10], and the effect of adding all the learnt clauses needed to solve the formula by Picosat [5]. Table 8 shows the results. The preprocessing has almost no impact on the modularity of the formula, except for α=1, because the preprocessing already solves the formula. With respect to the addition of learnt clauses, it is interesting to observe that

n	α	Orig.	Preproc.	Learnt	Connect. Comp.
300	1	0.459	0	0.453	1
300	2	0.291	0.235	0.291	1
300	4	0.190	0.188	0.073	1
300	4.25	0.183	0.182	0.041	1
300	4.5	0.177	0.177	0.045	1
300	6	0.150	0.150	0.120	1
300	10	0.112	0.112	0.171	1

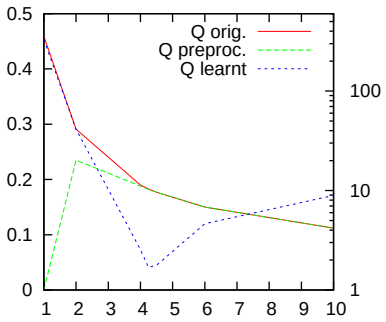

Table 8. Modularity (computed with GFA) of random formulas with 300 variables varying the clause variable ratio after and before preprocessing the formula with the Satelite preprocessor [10], and with all learnt clauses included.

in the peak transition region $\alpha = 4.25$, we get the lowest modularity. A possible explanation is that at the peak region we find the hardest instances, and in order to solve them the learnt clauses added by the solver tend to connect more communities. In [2] we observed that random SAT instances have not a scale-free structure[3], but that the addition of learnt clauses makes the formula clearly scale-free. On the contrary, we observe here that modularity tends to decrease with learning.

7 Conclusions

The research community on complex networks has developed techniques of analysis and algorithms that can be used by the SAT community to improve our knowledge about the structure of industrial SAT instances, and, as result, to improve the efficiency of SAT solvers.

In this paper we address the first systematic study of the community structure of SAT instances, finding a clear evidence of such structure in most analyzed instances. In fact, some features, like Moskewicz's activity-based heuristics, were already designed thinking on the existence of this kind of structure. Here we go a step further, and we propose the use of an algorithm that is able to compute the communities of a SAT instance. It verifies the assumption about the existence of this community structure. The algorithm could also be used directly by SAT solvers to focus their search.

References

1. Amir, E., McIlraith, S.A.: Partition-based logical reasoning for first-order and propositional theories. Artif. Intell. 162(1-2), 49–88 (2005)

[3] This is another structural quality of formulas that is being used by modern SAT solvers.

2. Ansótegui, C., Bonet, M.L., Levy, J.: On the structure of industrial SAT instances. In: CP. pp. 127–141 (2009)
3. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI. pp. 399–404 (2009)
4. Barber, M.J.: Modularity and community detection in bipartite networks. Phys. Rev. E 76(6), 066102 (2007)
5. Biere, A.: Picosat essentials. JSAT 4(2-4), 75–97 (2008)
6. Biere, A., Sinz, C.: Decomposing SAT problems into connected components. JSAT 2(1-4), 201–208 (2006)
7. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (2008)
8. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity – NP-completeness and beyond. Tech. rep., Faculty of Informatics, Universität Karlsruhe (TH), Tech. Rep. a 2006-19 (2006)
9. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E 70(6), 066111 (2004)
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: SAT'05. pp. 61–75 (2005)
11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'03). pp. 502–518 (2003)
12. Fortunato, S.: Community detection in graphs. Physics Reports 486(3-5), 75 – 174 (2010)
13. Gent, I.P., Hoos, H.H., Prosser, P., Walsh, T.: Morphing: Combining structure and randomness. In: AAAI/IAAI. pp. 654–660 (1999)
14. Gomes, C.P., Selman, B.: Problem structure in the presence of perturbations. In: AAAI/IAAI. pp. 221–226 (1997)
15. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98). pp. 431–437 (1998)
16. Hogg, T.: Refining the phase transition in combinatorial search. Artif. Intell. 81(1-2), 127–154 (1996)
17. Järvisalo, M., Niemelä, I.: The effect of structural branching on the efficiency of clause learning SAT solving: An experimental study. J. Algorithms 63, 90–113 (January 2008)
18. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC. pp. 530–535 (2001)
19. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E 69(6), 066133 (2004)
20. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(2), 026113 (2004)
21. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E 76(3), 036106 (Sep 2007)
22. Slater, A.: Modelling more realistic SAT problems. In: Australian Joint Conference on Artificial Intelligence. pp. 591–602 (2002)
23. Walsh, T.: Search in a small world. In: IJCAI. pp. 1172–1177 (1999)
24. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393(6684), 440–442 (June 1998)
25. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: IJCAI. pp. 1173–1178 (2003)