On the Modularity of Industrial SAT Instances

Carlos Ansótegui Jordi Levy
Universitat de Lleida (DIEI, UdL)
Artificial Intelligence Research Institute (IIIA, CSIC)

Abstract. Learning, re-starting and other techniques of modern SAT solvers have been shown efficient when solving SAT instances from industrial application. The ability to exploit the structure of these instances has been proposed as the responsible of such success. Here we study the modularity of some of these instances, used in the latest SAT competitions. Using a simple label propagation algorithm we show that the community structure of most of these SAT instances can be identified very efficiently. We also discuss how this structure may be used to speed up SAT solvers.

Keywords. Satisfiability, graph theory, modularity

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 [7,14]. 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 [21].

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 will help us understand the nature of SAT instances, and will contribute to further improve the efficiency of SAT solvers. Watts and Strogatz [20] introduce the notion of *small word*, the first model of complex networks, as an alternative to the classical random graph models. Walsh [19] analyzes the small word 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. [10,11] propose the use of *randomization* and *rapid restart* techniques to prevent solvers to fall on the long tail of such kinds of distributions.

The notion of structure has been addressed in previous work [10,12,9,13]. The closest work to our contribution is [18], but it uses the notion of modularity in a different

Research partially founded by the CICyT research projects TIN2010-20967-C04-01/03 and TIN2009-14704-C03-01 .

Corresponding Author: Departament d'Informàtica i Enginyeria Industrial, Jaume II 69, 25001 Lleida, Spain; E-mail: carlos@diei.udl.cat

sense. 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 [4], it is shown that many SAT instances can be decomposed into connected components, and how to handle them within a SAT solver. They also discuss how what they call *component structure* can be used to improve the performance of SAT solvers.

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* [15] 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 [14] 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.

1. Preliminaries

Given a set of Boolean variables , a *literal* is an expression of the form or . A *clause* of length is a disjunction of literals, . We say that is the size of , noted , and that , if contains the literal or . A *CNF formula* or *SAT instance* of length is a conjunction of clauses, .

An (undirected) graph is a pair where is a set of vertices and satisfies . This definition generalizes the classical notion of graph , where , by taking if and otherwise. The degree of a vertex is defined as . A bipartite graph is a tuple where .

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 results into 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 . In 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. SAT instances use to be simplified: there are no two occurrences of a literal (because is equivalent to) nor two occurrences of the same variable and opposite sign (because is a tautology). Therefore, in the CVIG model, edges have weight one.

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

Definition 2 (Clause-Variable Incidence Graph (CVIG)) *Given a SAT instance over the set of variables , its clause-variable incidence graph is a bipartite graph , with vertices the set of variables and the set of clauses, and weight function:*

2. Modularity in Large-Scale Graphs

To analyze the structure of a SAT instance we will use the notion of *modularity* introduced by [16]. 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 optimal 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 and a partition of its vertices, we define their* modularity *as*

We call the first term of this formula the inner edges fraction, *for short, and the second term the* expected inner edges fraction, *for short. Then, .*

The (optimal) modularity *of a graph is the maximal modularity, for any possible partition of its vertices.*

Since the and the of a graph are both in the range , and, for the partition given by a single community, both have value , the optimal modularity of graph will be in the range . A value of is considered as a high evidence of community structure in a graph.

There has not been an agreement on the definition of modularity for bipartite graphs. Here we will use the notion proposed by [2] that extends Newman and Girvan's definition by restricting the random graphs used in the computation of the IEFto be bipartite. In this definition, communities may contain vertices of and of .

Definition 4 (Modularity of a Bipartite Graph) *Given a graph and a partition of its vertices, we define their* modularity *as*

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 [8] for a survey in the field. The decision version of modularity maximization is NP-complete [5]. 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 two algorithms specially designed to deal with large-scale networks: the greedy algorithms for modularity optimization [15,6], and a label propagation-based algorithm [17].

The first algorithm for modularity maximization is a *greedy method* of Newman [15]. 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 [6] 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 , where is the depth of the dendogram (i.e. the number of joining steps), the number of edges and the number of vertices. They argue that may be approximated by , 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* proposed by [17] (see Figure 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.

Table 1. Computation of the modularity of the 2010 SAT Race instances, using the Label Propagation Algorithm. Time (in seconds) is the CPU time needed to propagate labels (excluding parsing the formula and constructing the graph). is the modularity. is the number of communities, and the fraction of vertices in the largest community. Iter is the number of iterations of the algorithm.

3. 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.

Our experiments have been run on a cluster with the following specifications. Operating System: Rocks Cluster 4.0.0 Linux 2.6.9, Processor: AMD Opteron 248 Processor, 2 GHz, Memory: 1 GB, and Compiler GCC 3.4.3.

```
Input:
Output:   a labelling for
for  do label[] := ; freq[] := 0 endfor
do
    ord := shuffle()
    changes := false
    for :=0 to  do
         := most_freq_label(ord[],neighbors(ord[]))
        changes := changes    freq[ord[]]
        label[ord[]] :=
        freq[ord[]] :=
    endfor
while changes
return label
function most_freq_label()
     :=label[]
    for  do
        freq[] :=
    Max :=  freq[] = freq[]
return random_choose(Max)
```

Figure 1. Label Propagation Algorithm. 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.

We have observed that all instances of the same family have a similar modularity. Therefore, in Table 1, we only show average values, being their standard deviation quite small (smaller that 10% of the mean in almost all cases).

As one could expect, the time needed to compute the modularity in the CVIG model is bigger than the time for the VIG model, since CVIG graphs are bigger. However, the number of iterations of the algorithm is also bigger in the CVIG model.

If we take as an evidence of community structure, we can conclude that most families are modular in the VIG model, and that all but one are modular in the CVIG model. Except for the bioinf family, the modularity is always bigger in the CVIG model than in the VIG model. In some families, like mizh, fuhs or nec, the modularity for VIG is meaningless, whereas the CVIG graph shows a clear community structure. It could be concluded that the loss of information, during the projection of the bipartite CVIG graph into the set of variables, 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. , will be nearly zero, and . Therefore, we have to take into account that using Barber's modularity definition for bipartite graphs, as we do, if vertex degrees are small, modularity can be quite big compared with Newman's modularity.

Table 2. Modularity of the graph generated by the formula containing the first 100 learned clauses, and all learned clauses. In the first column we show the modularity of the original formula.

Table 3. Modularity of the learned clauses that have been contributed to prove the unsatisfiability of the original formula. Like in Table 2 we show results for the first 100 learned clauses, and for all clauses.

Finally, we also report results on the number of communities () and the fraction of vertices belonging to the largest community (). If all communities have a similar size, then . In some cases, like palacios and mizh, we have . This means that the community structure corresponds to a big (or some) big central communities surrounded by a multitude of small communities. The existence of a big community implies an expected inner fraction close to one, hence a modularity close to zero.

We have also conducted a study of the modularity of random 3-CNF SAT instances. For the VIG model, we have observed that, with variables and clauses (i.e. around the phase transition point) the average modularity is , the average number of communities and largest size fraction . Therefore, almost surely the algorithm collapses into a single community. For under-constrained problems with variables and clauses, the average modularity is . In this case there is a largest community with of the vertices, on average, although there are also additional communities. For the CVIG model, in the phase transition point, we get , and . For variables and clauses, we get , and .

Notice that in the CVIG model for random 3-CNF formulas, using the same argument as above, i.e. assigning a distinct community to each variable, and to each clause the community of one of its variables, we get a graph partition with . Therefore, in these examples the label propagation algorithm does not always compute the maximal modularity, and tends to collapse all communities into a single one. However, analyzing the results for each one of the formulas independently, we observe that in some of them the algorithm is able to compute this maximal modularity. Therefore, it would make sense to run the algorithm, with distinct seeds, and take the partition with maximal modularity.

4. Modularity of the Learned 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? We think that the (indirect) exploitation of the community structure is responsible for the success of SAT solvers based on learning and restarting techniques. Thus, the second question is: how is this statement supported?

Let us try to answer the second question. If we do not exploit the community structure, in a pure DPLL schema, the best thing we can do is to assign the variables that occur more often in small clauses. Then, every time we assign we can remove all clauses con-

Velev instances are huge. We have to compute their modularity in a computer with more memory. In the case of the CVIG model, even this second computer is unable to compute the partition.

taining (because they become satisfied) and remove the literal from the clauses that contain it, possibly creating unary clauses that force the assignment of a variable. However, if most frequent variables belong to distinct communities, and most clauses are local to a community, we will loose a lot of time deciding about variables that do not contribute to falsify any clause. If we use learning, we can revert this situation. If most of the original clauses are local, since learned clauses are obtained by resolution from them, with high probability the learned clauses will be also local. When the solver learns after a conflict, the solver has already decided the assignment for all these variables, among others. I.e. the sequence of decision is something like

Deciding about the variables has been useless. Moreover, possibly we will repeat this work for the opposite assignment of those variables, that being in other community, have no influence in our conflict. However, now the solver can backtrack to the decision of , not reconsidering . Moreover, we can restart the execution, and if we use an activity-based heuristic, next time the solver will try to decide on the variables that have been participating in recent conflicts, i.e. with higher probability on the variables . We think that, this way, the solver centers its attention on the variables of a single community, or a few communities. Obviously, we could exploit the community structure directly, if previously we had computed a good partition. We can apply learning locally in each community, and re-start the algorithm with a lot of supplementary clauses.

To test the thesis that a considerable part of learning is performed locally inside one or a few communities, we generate formulas with the learned clauses and analyze their modularity. We use the picosat SAT solver [3] (version 846), since it incorporates a conflict directed clause learning algorithm, activity-based heuristics, and restarting strategies.

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 learned clauses. Since both graphs (the original formula and the learned 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 learned clauses. The partition classifies the variables of this second graph 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 learned clauses and for all the learned clauses. We also want to know the influence of the quality of the learned clauses. Therefore, we also repeat the experiment for all the learned clauses (Table 2), and only using the clauses that participate in the proof of

A clause is local, in the VIG model, if all its variables belong to the same community, and in the CVIG model, if it is only connected to variables of the same community.

unsatisfiability (Table 3). Notice that Table 3 contains fewer entries than Table 2 because we can only consider unsatisfiable instances. Notice also that picosat is not able to solve all 2010 SAT Race instances, therefore Tables 2 and 3 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 learned clauses, but equivalent results if we consider all learned clauses. There are not significant differences if we use all learned 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 learned 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 to adjust it to the modified formula.

It is worth to remark that, in Table 2, the average , for the VIG and the CVIG models and the first 100 learned clauses, is respectively and .

5. 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. We discuss how this structure is already exploited by modern SAT solvers. 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 propose 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 can also be used directly by SAT solvers to focus their search.

References

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

[2] M. J. Barber. Modularity and community detection in bipartite networks. *Phys. Rev. E*, 76(6):066102, 2007.

[3] A. Biere. Picosat essentials. *JSAT*, 4(2-4):75–97, 2008.

[4] A. Biere and C. Sinz. Decomposing sat problems into connected components. *JSAT*, 2(1-4):201–208, 2006.

[5] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity – NP-completeness and beyond. Technical report, Faculty of Informatics, Universität Karlsruhe (TH), Tech. Rep. a 2006-19, 2006.

[6] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, 2004.

[7] N. Eén and N. Sörensson. An extensible sat-solver. In *6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[8] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[9] I. P. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *AAAI/IAAI*, pages 654–660, 1999.

[10] C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *AAAI/IAAI*, pages 221–226, 1997.

[11] C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98)*, pages 431–437, 1998.

[12] T. Hogg. Refining the phase transition in combinatorial search. *Artif. Intell.*, 81(1-2):127–154, 1996.

[13] M. Järvisalo and I. Niemelä. The effect of structural branching on the efficiency of clause learning sat solving: An experimental study. *J. Algorithms*, 63:90–113, January 2008.

[14] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *DAC*, pages 530–535, 2001.

[15] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6):066133, 2004.

[16] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, 2004.

[17] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76(3):036106, Sep 2007.

[18] A. Slater. Modelling more realistic sat problems. In *Australian Joint Conference on Artificial Intelligence*, pages 591–602, 2002.

[19] T. Walsh. Search in a small world. In *IJCAI*, pages 1172–1177, 1999.

[20] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[21] R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.