SATGraf: Visualizing Community Structure in Boolean SAT Instances

Zack Newsham, William Lindsay, Jia Hui Liang,
Krzysztof Czarnecki, Sebastian Fischmeister, and Vijay Ganesh

University of Waterloo, Ontario, Canada

Abstract. In this paper, we present SATGraf, a tool for visualizing the *community structure* of the variable-incidence graph of a Boolean SAT formula. In addition, SATGraf has an evolution mode that allows one to visualize the dynamically evolving community structure of an input SAT instance as a CDCL SAT solver processes it. This tool enabled us to learn how the solver morphs the community structure of real-world SAT instances, and in turn, led us to some meaningful hypotheses about the connection between community structure of such instances and the running time of the CDCL SAT solver.

1 Introduction

In recent years, we have witnessed dramatic improvements in the efficiency of conflict-driven clause-learning (CDCL) Boolean SAT solvers [1–4]. There is general agreement in the broader solver community that CDCL SAT solvers somehow leverage structure inherent to real-world instances for better performance. In fact, some recent papers [5, 6] suggest that real-world SAT instances may have inherent *community structure*, and that may influence the running time of CDCL SAT solvers [7]. Informally, a community [5] in a SAT formula, when viewed as the variable-incidence graph[1], is a sub-graph that has more edges internal to itself than going out to the remainder of the graph.

Motivated by these discoveries [5–7], we built SATGraf, a visualization and evolution tool that displays the structure inherent to Boolean formulas and shows how this structure is morphed by modern CDCL SAT solvers as they solve these formulas. SATGraf takes as input a Boolean formula, constructs the corresponding variable-incidence graph, finds the inherent community structure, and displays it. SATGraf also shows how CDCL solvers morph the input formula, often solving it community by community.

While there are other tools that help us visualize the graph structure of SAT formulas [8], they do not display their community structure. A motivation for

[1] A variable-incidence graph of a Boolean SAT formula is one where the variables of the formula are nodes and there is an edge between two nodes if the corresponding variables occur in the same clause. In the rest of the paper we will not distinguish a formula from its variable-incidence graph.

us to build SATGraf was to get clues as to how CDCL solvers exploit structure, and indeed it led us to many falsifiable hypotheses that we subsequently experimentally tested (described in a related paper under submission [9]).

2 Background

To understand SATGraf and its effectiveness, it is important to understand the concept of communities. The idea of decomposing graphs into *n*atural communities arose in the study of complex networks. Informally, a network or graph is said to have community structure, if the graph can be decomposed into sub-graphs where the sub-graphs have more internal edges than outgoing edges. Each such sub-graph is called a community. Modularity is a measure of the quality of the community structure of a graph which ranges from 0 to 1 where 0 is a poor community structure and 1 a strong community structure. Modularity is often used in optimization methods for detecting community structure in networks. The precise definition and its calculation can be found in [5]. Many algorithms [10, 11] have been proposed to solve the problem of finding an optimal community structure of a graph, the most well-known among them being the one from Girvan and Newman [10]. There are many different ways of computing the modularity value and we refer the reader to these paper [10–12].

3 SATGraf Implementation

SATGraf is implemented in three phases:

Phase 1: First, it converts an input Boolean formula (represented in the standard DIMACS format) into its corresponding variable-incidence graph.

Phase 2: Second, it computes the community structure based on user's choice of either the Clauset-Newman-Moore (CNM) algorithm [10] or the online (OL) community algorithm [11]. We modified the CNM algorithm to make it more efficient, and it is the default community discovery algorithm in SATGraf. The original CNM and OL algorithms and our modifications are described in section 3.1.

Phase 3: Finally, SATGraf uses a modified version of the force-directed graph layout algorithm by Kamada and Kawai called the KK algorithm [13] to render the community structure. The original algorithm and our modifications are described in section 3.1.

Figure 1 shows the graph generated by SATGraf for two instances from the SAT 2013 competition [14]. The one on the left is an industrial instance, and the one on the right is a randomly-generated instance. Each distinct colour is mapped to a unique community to make it easy to discern the various communities. The colour black is reserved for inter-community edges. As is evident, the industrial instance has lot more distinct communities that can be neatly partitioned, while

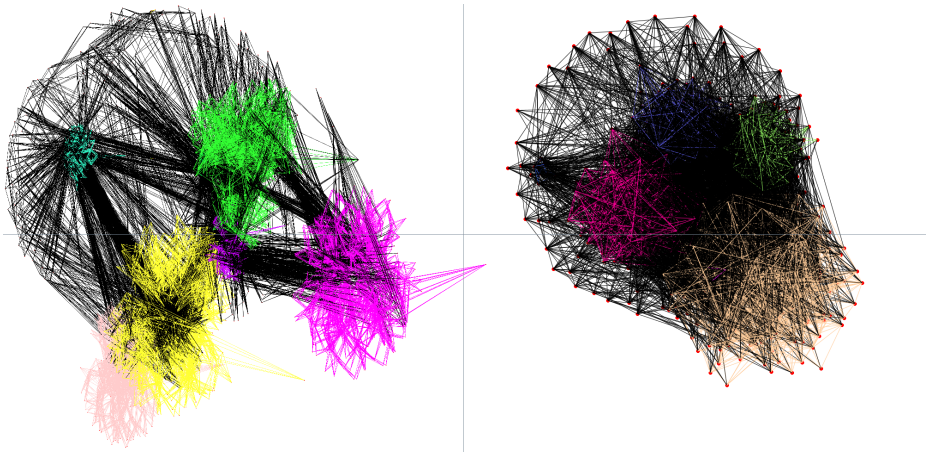

Fig. 1. Community structure of industrial instance *mrpp_4x4#4_16* (Left half), and of random instance *unif-k3-r4.267-v421-c1796-S4839562527790587617* (Right half). Both instances are from the SAT 2013 Competition.

the randomly generated instance has one big community that has lots of edges linking it tightly, with very few other discernible communities.

SATGraf presents the evolution of the formula by interacting with a modified version of MiniPure[2], which periodically generates a new graph of the SAT formula as it is being solved. The period is user specifiable and corresponds to the number of conflicts (n) discovered by MiniPure. The output from MiniPure after n conflicts is the formula with all satisfied clauses removed. The remaining clauses do include the learnt clauses generated by the solver. It is worth noting that the tool is not restricted to MiniPure.

3.1 Algorithms used in SATGraf

CNM Algorithm The purpose of the CNM algorithm [10, 15] is to identify communities in a graph. Initially every node is its own community. For every pair of communities, the pair that maximizes the modularity score is merged into a single community. This process is repeated until the modularity score cannot be maximized any further. We implemented the CNM algorithm with some very minor data structure modifications.

OL Algorithm The online community detection algorithm by Zhang et al. [11] has the same purpose as the CNM algorithm and uses the same metric - modularity - to determine the quality of the community structure. However, the difference between the two methods is that the online method has a linear worstcase time complexity in the number of edges, compared to that of the CNM algorithm which is worstcase exponential in the number of edges. For real-world examples this led to 600-fold speedup [11]. The only drawback to the algorithm is that the

modularity found by the algorithm, in our experience, is strictly worse than the modularity found by the CNM algorithm. Consequently, the graphs' communities will not be as strongly connected and may vary each time the algorithm is ran.

KK Algorithm The Kamada-Kawai algorithm [13] is a force-directed graph drawing algorithm for general undirected graphs. It assigns "forces" to edges and nodes based on their position relative to each other and then calculates the *repulsive and attractive forces* between them to reposition them. The KK algorithm's complete explanation can be found in this paper [13]. Several small modifications were made in our implementation of the published algorithm. Firstly, the KK algorithm requires that graphs be fully connected. Unfortunately, this is not always the case for SAT instances. Hence, we implemented a system to create *proxy* edges between nodes for the purpose of the layout. These are removed after the position of each node is determined. The second, and more important modification was to implement a wrapper for the KK algorithm that takes advantage of the community structure present in most large SAT instances to reduce the complexity of the algorithm. Rather than submitting an entire graph to the script for processing, each community is submitted in turn. These communities are then laid out relative to each other using an approximation that replaces each community by a hypernode in the graph. Once each hypernode's position is determined, the communities are brought back and their positions are scaled relative to each other's size. This results in a decrease in the algorithm's complexity due to the far smaller number of communities versus nodes (usually less than 200 communities are present). This yielded a performance improvement by a factor of three.

4 Results

SATGraf has been tested on several industrial, hard combinatorial, and randomly-generated formulas from the 2013 SAT competition[14]. The time taken to display the community structure of a single instance grows with the size of the input formula. This is to be expected due to the nature of the community detection and placement algorithms. The resulting images using the CNM community detection and KK layout algorithms, can be seen in Figure 1. As mentioned earlier, the right half of the Figure 1 is a radomly generated SAT instance from the 2013 SAT competition, whereas the left half of the Figure 1 is the graph of an industrial SAT instance from the same competition. The community structure of the industrial instance has much better modularity than the one for the randomly-generated instance. This can be verified both visually and through the modularity measure of quality of the community structure: the industrial instance has a modularity of 0.437204, while the randomly-generated one has a modularity of 0.132647. Their solve times using MiniPure are also different; The industrial instance takes 0.076 seconds to solve compared to the randomly-generated instance which times out after 5000 seconds. This suggests that the

community structure of a SAT formula might impact the running time of the SAT solver.

SATGraf's evolution feature is partly shown in two pictures in Figure 2. The SAT instance here is obtained from a feature model [16] called *Fiasco* that can be downloaded from SATGraf website [17]. The entire evolution of *Fiasco* can be found at [17]. We chose this SAT formula since it is a good representative of an industrial application of SAT solvers. Furthermore, this instance is small enough so that we can actually show, in a timely manner, how the SAT solver dynamically morphs its graph (the instance and the generated learnt clauses). Furthermore, the MiniPure solver [2] solved this formula without generating too many conflicts, and thus it was easier to make sense of the evolution of the graph of this instance. Observing the evolution showed an interesting trend, namely, the removal of entire communities during the solving process. This evolution can be seen when going from the graph of the original SAT formula in the left half of Figure 2, to the graph after MiniPure generates the first conflict shown in the right half of Figure 2. It is easy to see that some of the communities have completely disappeared by the abscence of their associated colour, i.e., the corresponding clauses have been satisfied.

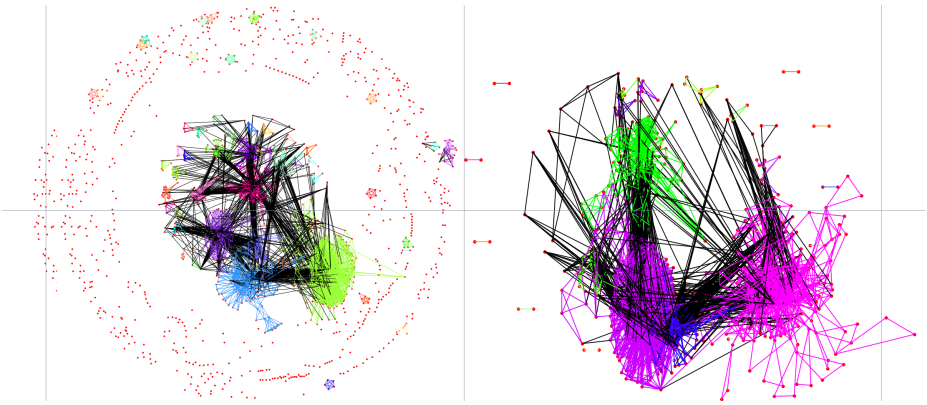

Fig. 2. Partial evolution of the *Fiasco* problem

5 Related Work

Carlos Ansotegui et al. [5] were the first to discuss the presence of community structure in SAT formulas. The idea of community structure arose in the context of complex networks, and many algorithms have been proposed to discover them [18, 10]. Below we give a brief description of other tools that provide visualization and/or evolution capabilities and highlight the differences between these tools and SATGraf.

SATGraf is the only tool that we know of that has both visualization capabilities to view the community structure of SAT instances, and evolution feature that shows how the community structure is morphed by a CDCL SAT solver as it solves an input instance. While other tools [8, 19–21] have visualization and/or evolution capabilities, they do not focus on the community structure of SAT instances nor do they show how the solver morphs the community structure of input SAT instances. Instead these tools allow the user to view the SAT instance as a graph without any community structure information. The following Table 1 highlights the differences between the various visualization tools that we found. Those differences range accross a handful of categories such as interactive (ability to set a value to a variable on the graph), evolution (ability to see the evolution of the SAT formula), community (ability to display the community structure), 3D (three dimensional capability), and implication (can generate the implication graph).

DPViz [8], probably the closest to SATGraf in terms of features, is a graphing tool designed to expose how a CDCL solver morphs a SAT instance as it is being solved. It offers a number of features such as multiple layout algorithms, zooming into the graph to show more detail, the ability to set specific values on literals displayed in the graph, and performing unit propagation.

Tool	Interactive	Evolution	Community	3D	Implication
DPViz[8]	✓	✓	✗	✗	✓
GraphInsight[20]	✓	✗	✗	✓	✗
iSat[19]	✗	✓	✗	✗	✗
GraphViz[21]	✗	✗	✗	✗	✗
SATGraf	✓	✓	✓	✗	✓

Table 1. Comparison of Tools

Each tool presented above has different strengths and weaknesses. However, the only tool that can accomplish visualizing community structure of a SAT formula, both in its original state and while being solved by a SAT solver, is SATGraf.

6 Conclusion

SATGraf presents a way to visualize a SAT instance's community structure. Furthermore, SATGraf has the ability to dynamically graph the community structure of a CDCL SAT solver's progress while solving a SAT formula. These features were shown to be unique to SATGraf when compared to various similar tools. These new capabilities yielded hypotheses relating the quality of the community structure with the performance of a CDCL SAT solver. We found that the better the modularity is, the less time the SAT solver needs, and the CDCL SAT solver seems to solve SAT formulas by solving a community at a time.

References

1. Niklas Een and Niklas Sörensson. Minisat: A SAT solver with conflict-clause minimization. *SAT*, 5, 2005.
2. T. Taiwan and H. Wang. Minipure, 2013. http://edacc4.informatik.uni-ulm.de/SC13/solver-description-download/134, last viewed January 2014.
3. Gilles Audemard and Laurent Simon. Glucose: a solver that predicts learnt clauses quality. 2009.
4. A Biere. Lingeling. *SAT Race*, 2010.
5. C. Ansotegui, J. Giraldez-Cru, and J. Levy. The community structure of SAT formulas. In *Theory and Application of Satisfiability Testing - SAT 2012*, pages 410–423. Springer, 2012.
6. C. Ansotegui and J. Levy. On the modularity of industrial SAT instances. In *Frontiers in Artificial Intelligence and Applications*, volume 232, pages 11–20. Springer, 2011.
7. Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, and Jordi Levy. The fractal dimension of SAT formulas. *CoRR*, abs/1308.5046, 2013.
8. Carsten Sinz and Edda-Maria Dieringer. DPvis–a tool to visualize the structure of SAT instances. In *Theory and Applications of Satisfiability Testing*, pages 257–268. Springer, 2005.
9. Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Impact of community structure on SAT solver performance. In *Under Submission*. 2014.
10. Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.
11. Wangsheng Zhang, Gang Pan, Zhaohui Wu, and Shijian Li. Online community detection for large complex networks. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1903–1909. AAAI Press, 2013.
12. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks, 2003. http://arxiv.org/pdf/cond-mat/0308217.pdf, last viewed December 2013.
13. Tomihisa Kamada and Satoru Kawai. A general framework for visualizing abstract objects and relations. *ACM Trans. Graph.*, 10(1):1–39, January 1991.
14. SAT competition 2013, 2013. http://satcompetition.org/2013/, last viewed January 2014.
15. J. Leskovec. Snap system. http://snap.stanford.edu/snap/index.html, last viewed January 2014.
16. Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990.
17. Z. Newsham, W. Lindsay, J. Liang, K. Czarnecki, S. Fischmeister, and V. Ganesh. Evograph: Results, 2014. http://ece.uwaterloo.ca/∼vganesh/EvoGraph/Results.html, last viewed January 2014.
18. S. Fortunato, V. Latora, and M. Marchiori. Method to find community structures based on information centrality, 2004. http://www.w3.org/People/Massimo/papers/2004/community_pre_04.pdf.
19. Ezequiel Orbe, Carlos Areces, and Gabriel Infante-López. isat: structure visualization for SAT problems. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 335–342. Springer, 2012.

20. Carlo Nicolini and Michele Dallachiesa. Graphinsight: An interactive visualization system for graph data exploration. http://www.graphinsight.com.
21. A. Bilgin, J. Ellson, E. Gansner, O. Smyrna, Y. Hu, and S. North. Graphviz - graph visualization software. http://www.graphviz.org/.