On the community structure of SAT-BMC problems

Xavier Gillard and Charles Pecheur

Université Catholique de Louvain, Louvain-la-Neuve, Belgium
{xavier.gillard,charles.pecheur}@uclouvain.be

Abstract

A common belief says that modern SAT solvers are efficient because of their ability to exploit structural properties of industrial problems. However, we only have a limited understanding of what is covered by this 'structure'. A recent hypothesis suggests the community structure as a candidate definition for that notion. Our paper proposes a tool helping to understand community structure of SAT instances generated by BMC.

1 Introduction

It is commonly believed in the SAT community that the efficiency of modern SAT solvers stems from their ability to exploit the structure of the original problem. However, that notion of structure is often ill defined and very little is known about the reason solvers perform so well in practice. A recent line of work on SAT suggests that community structure is a good candidate for that 'structure' definition [2, 22, 23]. In that context, community structure is a property reflecting the possibility to split a SAT instance in almost disjoint subproblems. Indeed, the modularity measuring that structure was shown to be strongly correlated with the execution time of CDCL solvers [22]. However, it was proved [21] that some pseudo-industrial instances with a good community structure [2] require an exponential time for all resolution-based solvers. Such instances are therefore hard even for the most sophisticated CDCL solvers. Hence, it is believed that industrial instances exhibit additional features which are not fully explained by the community structure. In this paper we present a tool we developed to help understanding the community structure of industrial SAT problems. In particular, we focus on instances generated by SAT-based Bounded Model Checking (BMC) [9], a well established formal verification method based on SAT. Our paper is structured as follows: we begin with a brief introduction to community structure, modern SAT solvers and BMC. Then we present our tool and discuss some early observations we made. Finally, we propose research perspectives based on our prospective results.

2 Background

Community structure. Informally, the community structure of a graph is a structural property of that graph that decomposes it into *almost disjoint* sets of nodes. More precisely, these sets of nodes are called *communities* and consist of a cluster of nodes that are densely connected with one another and sparsely with the rest of the graph. There exist efficient algorithms capable of decomposing a graph into communities (e.g. the Louvain Method [11]).

When it comes to analyzing an instance of SAT problem, the community structure is generally interpreted over the variable incidence graph (VIG) derived from the boolean formula encoding the problem. Such a VIG is built from the CNF representation of the formula by having one vertex per *variable* of the problem and one edge between two variables if they appear together in one same clause (irrespective of their literal polarity).

Modern SAT solvers. Modern SAT solvers such as MiniSat [16] or Glucose [7] implement a Conflict Driven Clause Learning approach. These build upon the traditional DPLL [15] algorithm to implement a resolution refutation proof system, enhanced with the ability to learn new (redundant) clauses. All these additional clauses are guaranteed to derive from the original problem and are learned by the solver upon conflicts [26]. Additionally, CDCL solvers enrich their resolution algorithm with a set of techniques and heuristics aiming at maximizing the information gained from the learning process [28, 20, 24, 19, 5, 4, 6, 3, 10], which contributes to further pushing the limits of the set of tractable SAT instances.

Meanwhile, the question of why CDCL solvers work so well in practice remains largely unanswered. The traditional approach consists in a proof-complexity-theoretic argument saying that *because* of its capability to learn and remember conflict clauses, CDCL implements a proof system (General Resolution Refutation) that is up to exponentially more efficient than that of DPLL (Tree Like Resolution Refutation) [8, 18, 24]. However, this argument is of no use in understanding the rationale behind the good performance of such solvers on *industrial problems*[1]. Thus, it is widely believed that CDCL solvers are able to exploit some kind of structure of the underlying problem. Hence the recent interest in the community structure of industrial problems.

Bounded Model Checking (BMC). Model checking consists in verifying that a temporal formula ϕ expressed in LTL holds for all the traces of a model M. Bounded Model Checking [9] limits to traces shorter than k (written $M \models_k \phi$). To do so, a BMC model checker tries to disprove $M \models_k \phi$ by finding a trace of length k violating the property ϕ. Concretely, the refutation process is done symbolically by means of a reduction to SAT. In other words, a boolean formula $[\![M, \neg\phi]\!]_k$, which is satisfiable iff there exists a violating trace, is fed to some SAT solver. The solution of that satisfiability test determines whether or not $M \models_k \phi$ and yields an appropriate counter-example if one exists. For our matter, the key element to understand about the generation of $[\![M, \neg\phi]\!]_k$ is that it is generated as a sequence of k "blocks of variables" encoding the possible values of the k states of a bounded trace of M.

3 Experiments

A community analysis tool. The tool we developed[2] in order to analyze the community structure of SAT-BMC problem instances builds upon the PyNuSMV framework which has recently been extended to expose SAT capabilities of NuSMV [17]. Thanks to that framework, our tool is able to generate a series of SAT problem instances[3] corresponding to the various formulas submitted to a solver when performing a bounded verification of some given SMV model. Additionally, it uses Igraph [13] to build, manipulate and visualize the VIG associated with each of the aforementioned SAT problems. More interestingly, our tool combines the features of the two libraries to compute the graph modularity, detect the community structure of the VIG and then reconcile the variables with their semantic meaning in the original SMV model. This permits the use of data mining algorithms to detect patterns of semantic information frequently associated in the various communities. To that end, we used the algorithms implemented in Pymining [14] to detect frequent patterns (with relim) and frequent sequences. The

[1] To explain their lack of efficiency on *random instances*, it has been proved that random k-CNF formulas have exponential resolution complexity [1, 2, 12]. Obviously, that tells nothing about the efficiency of CDCL on industrial instances which is our main focus.

[2] Technical details are found on the project page http://bit.ly/2vWr8bM

[3] And dump them to DIMACS CNF format for later analysis if necessary.

information learned that way suggests what the hidden meaning of the communities could be. Moreover, our tool is also able to collect and plot statistics about the succession of generated instances. The combination of all the above features provides a fairly complete and efficient toolkit to analyze the structure of BMC instances. However, our tool slightly suffers from its inability to associate a semantic meaning to the auxiliary variables introduced by the Tseitin transformation [27] used to encode the raw SAT-BMC problem in CNF.

Early results. So far, a full scale empirical analysis of the community structure of BMC instances has not been realized yet. We have analyzed two small models (dining philosophers and a diagnosability) and one real industrial problem (a railway interlocking). We observed that communities tend to represent either the evolution of an object (ie philosopher, fork) over a certain period of time (time frame 1, 2 and 3) or the synchronization of the model at some point of time (all the forks in the 2nd time frame). Surprisingly though, we observed that often the communities tend to group variables from three consecutive time blocks while we would have expected no more than two to reflect the application of the Kripke Structure's transition relation. We haven't yet found any convincing reason explaining that observation, which remains to be confirmed by further experiments.

We also observed that the modularity of the SAT instances generated for known hard problems increases for small values of k and diminishes for higher bounds, which corroborates the results of [22]. Besides that, we also noted that the modularity of SAT instances generated for problems comprising a model only (formula ϕ is vacuously true) seems to saturate. We believe that this phenomenon might be related to the *diameter*[4] of the model.

4 Related work

Analyzing the community structure of industrial SAT instances has only gained interest very recently and is still actively investigated. Therefore, a tool called SatGraf [23] has been developed to analyze the impact of that structure on the CDCL runtime. Even though it shares some of its feature with the tool presented here, SatGraf does not consider the semantic level of the problem. A reflection somewhat similar in its essence to the one presented here has been held by Shishmarev et al. in [25]; although it took place in a completely different setup than the current one. Indeed, their focus wasn't set on the community structure but solely on getting an understanding of the meaning of learned clauses in the context of constraint programming. Hence, their results cannot be lifted to BMC.

5 Perspectives and conclusion

Encouraged by the above early results, we believe that our tool might greatly help in gaining a better understanding of the community structure that arises from SAT-BMC problems. This objective is a necessary step that could help designing BMC variants that best exploit the capabilities of current SAT solvers. Indeed, having a proper understanding of model features that give rise to the communities might serve the point of designing a preprocessor exploiting these features to generate redundant clauses, thereby easing the resolution without having to go through the lengthy conflict-learning process. Additionally, this kind of knowledge might help to scale SAT-BMC horizontally with the design of a semantically grounded heuristic to split the problem before passing it on to a parallel SAT solver.

[4] The smallest k to to ensure that $M \models_k \phi \iff M \models \phi$.

References

[1] Dimitris Achlioptas. Random satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of satisfiability*, chapter 8, pages 245–270. IOS press, 2009.

[2] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of SAT formulas. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 410–423. Springer, 2012.

[3] Gilles Audemard, Benoît Hoessen, Said Jabbour, Jean-Marie Lagniez, and Cédric Piette. Penelope, a parallel clause-freezer solver. In *SAT Challenge 2012: Solver and Benchmarks Descriptions*, pages 43–44, 2012.

[4] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Saïs. On freezing and reactivating learnt clauses. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 188–200. Springer, 2011.

[5] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, volume 9, pages 399–404, 2009.

[6] Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Principles and Practice of Constraint Programming*, pages 118–126. Springer, 2012.

[7] Gilles Audemard and Laurent Simon. Glucose and syrup in the SAT race 2015. *SAT Race*, 2015.

[8] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[9] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[10] Armin Biere and Andreas Fröhlich. Evaluating CDCL restart schemes. In *Proceedings POS-15. Sixth Pragmatics of SAT workshop*, 2015.

[11] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[12] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM (JACM)*, 35(4):759–768, 1988.

[13] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[14] Barthelemy Dagenais. Pymining, a few data mining algorithms in pure python. https://github.com/bartdag/pymining, 2015.

[15] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[16] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[17] Xavier Gillard. Adding SAT-based model checking to the pynusmv framework. Master's thesis, M.Sc. Thesis, Université Catholique de Louvain, 2016.

[18] Carla P Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3:89–134, 2008.

[19] Jinbo Huang et al. The effect of restarts on the efficiency of clause learning. In *IJCAI*, volume 7, pages 2318–2323, 2007.

[20] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[21] Nathan Mull, Daniel J Fremont, and Sanjit A Seshia. On the hardness of SAT with community structure. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 141–159. Springer, 2016.

[22] Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Impact of community structure on SAT solver performance. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 252–268. Springer, 2014.

[23] Zack Newsham, William Lindsay, Vijay Ganesh, Jia Hui Liang, Sebastian Fischmeister, and Krzysztof Czarnecki. Satgraf: Visualizing the evolution of SAT formula structure in solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 62–70. Springer, 2015.

[24] Lawrence Ryan. *Efficient algorithms for clause-learning SAT solvers*. PhD thesis, Citeseer, 2004.

[25] Maxim Shishmarev, Christopher Mears, Guido Tack, and Maria Garcia de la Banda. Learning from learning solvers. In *International Conference on Principles and Practice of Constraint Programming*, pages 455–472. Springer, 2016.

[26] João P Marques Silva and Karem A Sakallah. GRASP – a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997.

[27] Grigori Samuilovich Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constrained Mathematics and Mathematical Logic*, 1968.

[28] Hantao Zhang and Mark Stickel. Implementing the davis–putnam method. *Journal of Automated Reasoning*, 24(1-2):277–296, 2000.