Visualizing the Internal Structure of SAT Instances (Preliminary Report)

Carsten Sinz

Symbolic Computation Group, WSI for Computer Science
University of Tübingen, 72076 Tübingen, Germany
sinz@informatik.uni-tuebingen.de

Abstract. Modern algorithms for the SAT problem reveal an almost tractable behavior on "real-world" instances. This is frequently contributed to the fact that these instances possess an internal "structure" that hard problem instances do not exhibit. However, little is known about this internal structure. We therefore propose a visualization of the instance's variable interaction graph (and of its dynamic change during a run of a SAT-solver) as a first step of an empirical research program to analyze the problem structure. We present first results of such an analysis on instances of bounded model checking benchmark problems.

1 Introduction

Progress in SAT-checking over the last years has been tremendous. Problems that were completely out of reach ten years ago, can now be handled with success. Especially in hardware verification, SAT solvers (and the associated method of bounded model checking [1]) wrote a glittering success story, and mainly replaced the until then predominant BDD-based model checkers [2]. But also in other combinatorial problem domains, like planning [3], configuration [4], or software verification [5] SAT checkers were able to play out their strengths.

Although the advent of new methods and optimized implementations made SAT-solvers very successful on these "real-world" instances, large classes of SAT instances have remained (e.g., random 3-SAT problems with a clause-variable-ratio near the phase transition point) on which these solvers miserably fail—even on instances that are considerably smaller (by four to five orders of magnitude). Taken on its own, this fact is not very surprising, as SAT is an NP-complete problem. However, the reasons for this phenomenon are not very deeply understood, and—besides of rare and artificial cases—there is no a priori criterion available to decide whether a problem instance will behave benign or malicious.

The standard argument found in the literature to explain this dichotomy is that real-world instances are equipped with some kind of internal (and sometimes hidden) "structure" that makes these problems tractable. The term "structure", due to its vagueness, leaves much room for interpretation, though, and it remains unclear how this structure manifests itself and how it could be exploited. Among the methods that were proposed are randomization and clause learning, techniques that can be found in most implementations of modern SAT-solvers for real-world instances today [6, 7]. Concepts explaining the boundary between tractability and intractability include backbone variables [8] and backdoor sets [9]. Although highly valuable from both an epistemological and a practical point of view, these concepts do not deliver an a priori criterion to directly "read off" the computational hardness of a given instance.

We therefore propose a novel, partly empirical approach in order to shed some light on an instance's internal structure. A major ingredient of our method is the visualization of internal variable dependencies as they emerge from the problem's *interaction graph* [10–12]. In this graph, propositional variables appear as nodes, and an edge is drawn between two nodes, if the edge's adjacent nodes' variables appear together in at least one clause of the problem instance. We propose to use these diagrams as auxiliary devices to support the hypothesis building process.

In the next sections we show first steps into this direction. Experiments were mainly conducted with examples from bounded model checking.

2 Experiments

We started our examination with an instance from bounded model checking that stems from equivalence checking of a 16-bit sequential shift-and-add multiplier with a combinatorial multiplier (see [13] for further information). We used the file `longmult8`, representing equivalence of bit 8 of the two circuit

designs.[1] All diagrams were produced with the freely available graph editor and layout program yEd (http://www.yworks.com).

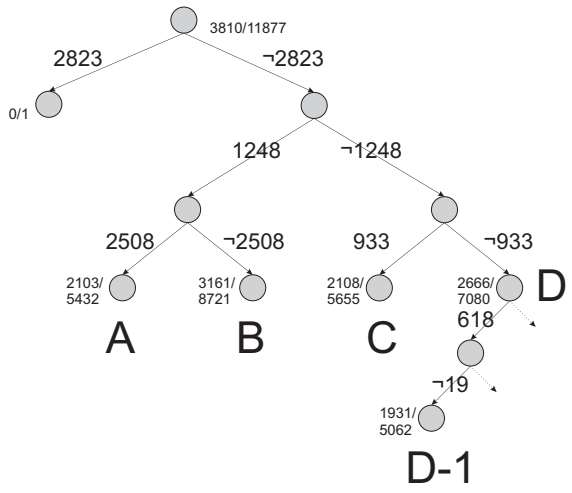

Fig. 1. Search tree of depth-bounded run of the DP algorithm on the `longmult8` benchmark problem.

It turned out that visualizing only the structure of the whole problem is not sufficient for explaination of structural phenomena. Therefore we simulated depth-bounded runs of a typical Davis-Putnam algorithm on these examples. The search tree of such a depth-bounded run on the `longmult8` instance is shown in Fig. 1. The tree shows the search tree up to depth 3, with the initial instance as root node of the tree. Each node is labeled with two numbers (n/k), where n indicates the number of different propositional variables of the instance and k the number of clauses. Child nodes are generated by case distinction (on the indicated variable) and subsequent unit propagation. On search depth 3, we have four non-trivial clause sets (A, B, C, D) with between 2103 and 3161 variables. To illuminate the dynamics of the interaction graphs, we have added a further node at an increased search depth of 5, named D-1. Interaction graphs for these five snapshots are shown in Figs. 2 and 3. iagram A-1 in the middle of Fig.3 additionally shows a magnified view of the top left part of Diagram A.

In these diagrams we used a refinement of ordinary interaction graphs, in which 2-clauses (aka 2-literal-clauses) are treated specially. Each 2-clause is represented as a directed or undirected arc in the graph, where we make the following distinction:

1. A clause with two positive literals is shown as a red, double-ended arrow.
2. A clause with one positive and one negative literal, say $(\neg x \vee y)$, is written as a blue arrow from variable x to variable y.
3. A clause with two negative literals is written as a green, double-ended arrow.

All other clauses are displayed with black edges between the involved variables.[2] For comparison reasons, we initiated experiments with other SAT instances. We used an instance from automotive product configuration[3], one instance of the well-known pigeon hole problems (`hole10`) and a random 3-SAT formula with 100 variables and 425 clauses (generated with SATO3.0). The last two instances are known to be hard for resolution-based SAT solvers, whereas the configuration instance is known to be very easy. The interaction graphs shown on the left of Fig. 4 correspond to a state during the run of a DP algorithm, where a few literals already have been fixed. After setting of three further variables and subsequent unit propagation the interaction graphs shown on the right resulted.

3 Observations and Hypotheses

During our experiments we made the following observations:

[1] http://www.cs.cmu.edu/~modelcheck/bmc/bmc-benchmarks.html
[2] Color versions of the diagrams of this article can be found at http://www-sr.informatik.uni-tuebingen.de/~sinz.
[3] File `C202_FW`, downloadable at http://www-sr.informatik.uni-tuebingen.de/~sinz/DC.

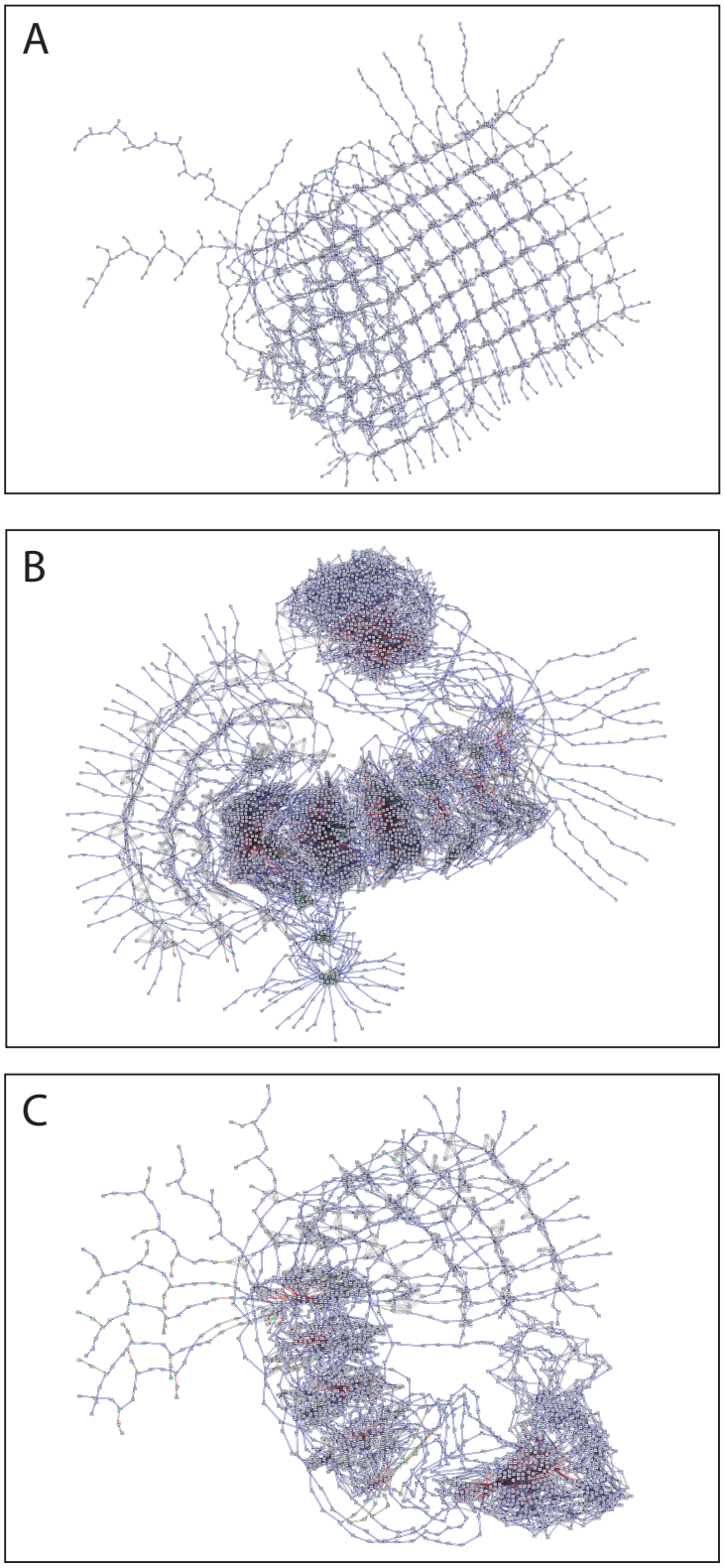

Fig. 2. Interaction graphs for three subinstances of `longmult8`, obtained by setting proposition 2823 to false, propositions 1248, 2508 and 933 to different truth values (as indicated in Fig. 1), and subsequent unit propagation.

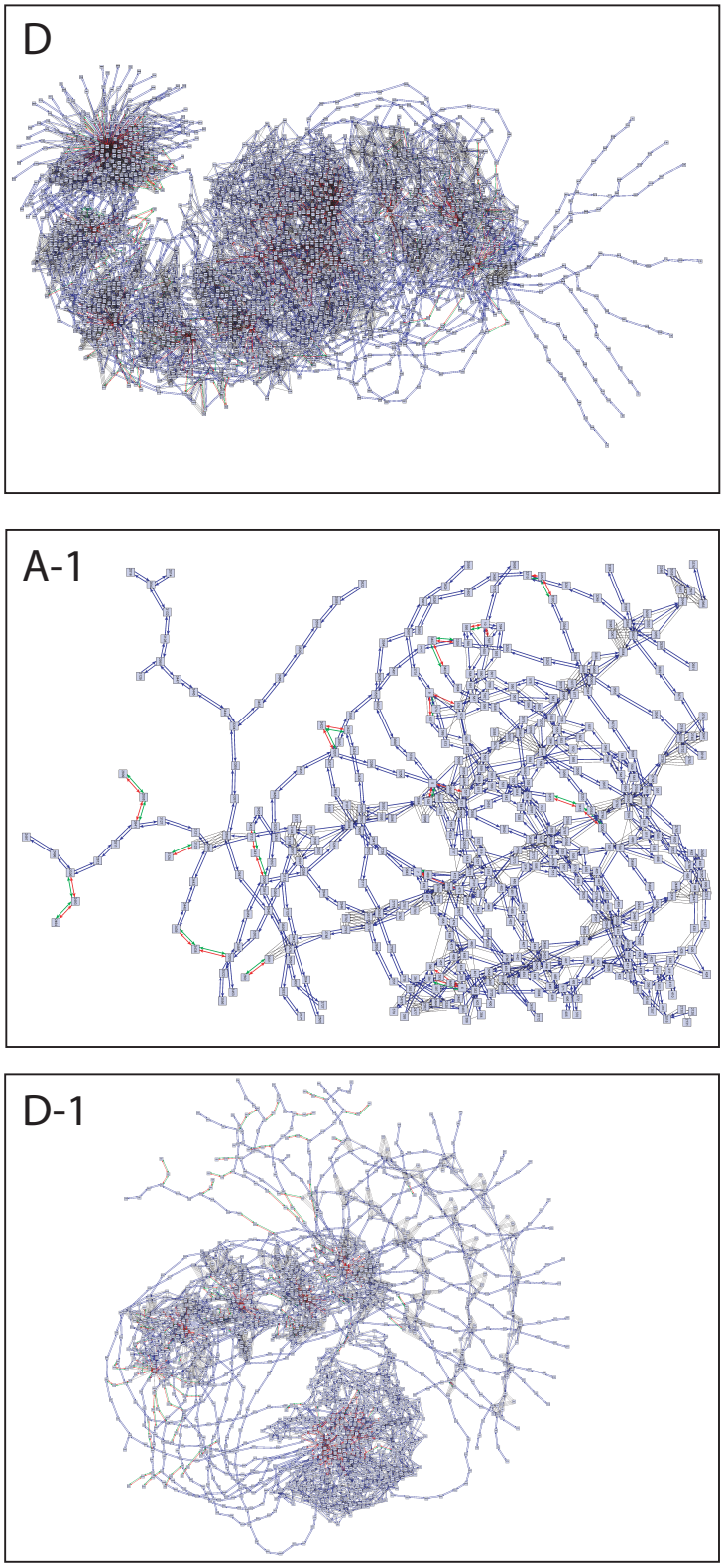

Fig. 3. On top: Interaction graph of the subinstance of `longmult8` obtained by setting propositions 2823, 1248 and 933 to false, and subsequent unit propagation. Middle and bottom: Two further interaction graphs: A-1 displays a magnified view of the top left part of Diagram A of Fig. 2, and D-1 shows the instance obtained from D by additionally setting proposition 618 to true, proposition 19 to false, and subsequent unit propagation.

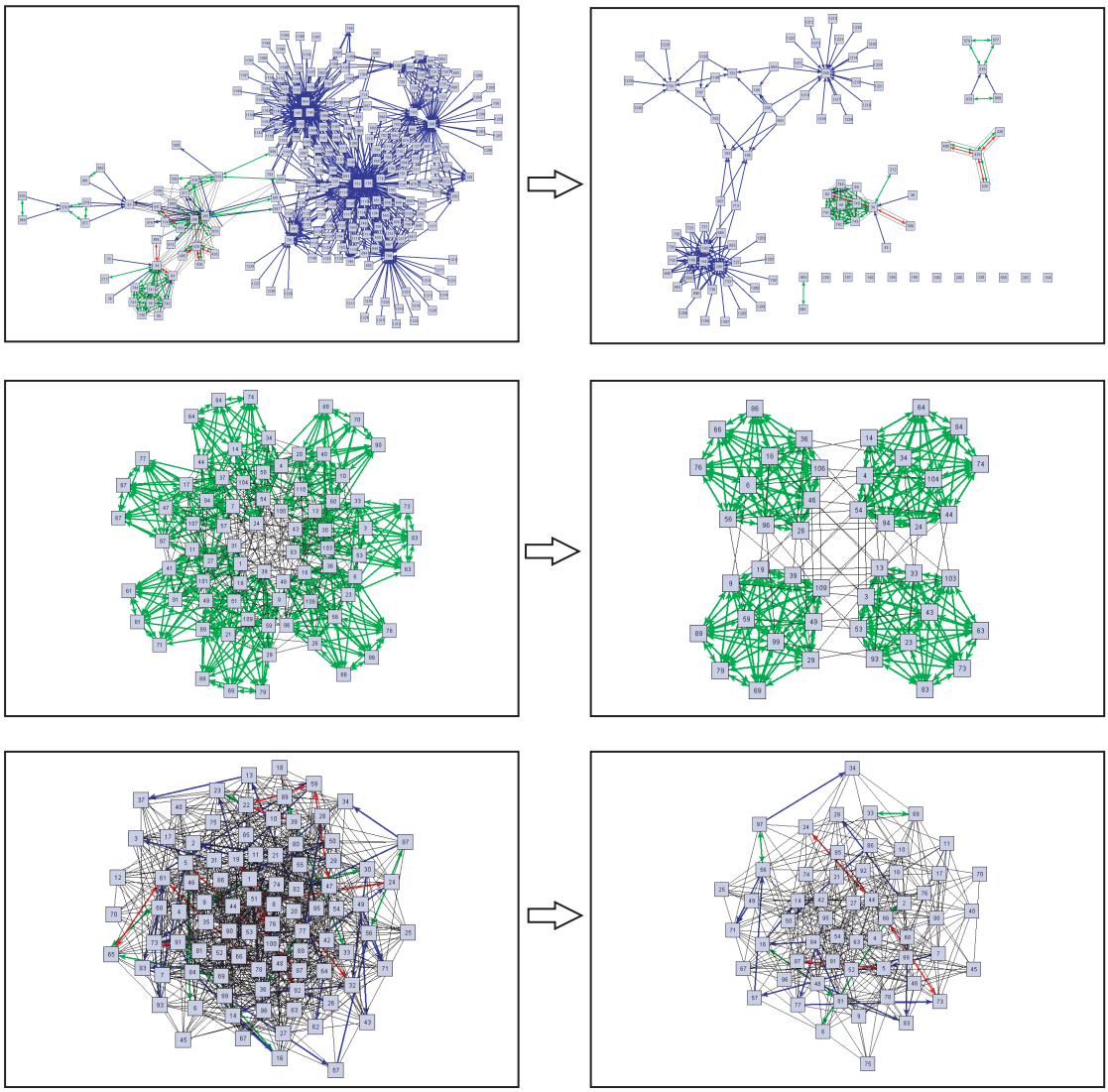

Fig. 4. Interaction graphs for different SAT instances before (left) and after (right) three steps of a DP algorithm run (and subsequent simplification by unit propagation). On the top, an instance from automotive product configuration is shown, in the middle a pigeon hole formula, and on the bottom a random 3-SAT formula with a clause-variable-ration near the phase-transition point.

1. On the `longmult8` instance we could observe noticeably many implication chains of considerable length (this can be seen best in Diagram A-1). In most cases, these chains were even made up of equivalences. However, most of them do not directly occur in the original instance, but only after setting of a few variables. (This may be typical for problem instances arising from bounded model checking.)
2. The internal structure as indicated by the interaction graphs can vary considerably between different parts of the search tree (compare, e.g. Diagram A and B).
3. Decomposition into independent subproblems may naturally occur in a DP run on a real-world instance (as in the configuration example in Fig. 4), but the decomposition property may equally well be missing (as in the `longmult8` example).
4. Hard problem instances (pigeon hole, random 3-SAT) not only possess a lower rate of variable reduction by unit propagation, but also lack the decomposition feature that appeared in the configuration instance. The hard instances seem to possess a "fractal" or "self-similar" behavior on problem reductions by unit propagation.

From these observations we (preliminarily) derive the following hypotheses:

1. Both long implication chains and decomposition into independent subproblems may explain the benign behavior of real-world instances. These are properties that we did not observe in hard instances.
2. Hard problems seem to exhibit a "fractal" or "self-similar" behavior on reductions by case-splitting and unit-propagation.

Whereas it is not evident how to make use of the second hypothesis, this seems not to be the case for the first one. Thus, we dare to hope for both theoretical and practical progress based on this observation. E.g., it may be worthwhile to develop specialized algorithms for instances that are supposed to decompose into smaller parts. Similarly, it may be profitable to directly optimize existing algorithms to make maximal use of long implication chains. For the future, we could also imagine to extend our analysis of interaction graphs to other concepts like scale-freeness, clustering, or betweenness centrality.

References

1. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19** (2001) 7–34
2. Velev, M., Bryant, R.: Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. J. Symb. Comput. **35** (2003) 73–106
3. Kautz, H., Selman, B.: Planning as satisfiability. In: Proc. 10th Europ. Conf. on Artificial Intelligence (ECAI'92), John Wiley and Sons (1992) 359–363
4. Küchlin, W., Sinz, C.: Proving consistency assertions for automotive product data management. J. Automated Reasoning **24** (2000) 145–163
5. Vaziri, M., Jackson, D.: Checking properties of heap-manipulating procedures with a constraint solver. In: Proc. 9th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03). Number 2619 in LNCS, Warsaw, Poland, Springer (2003) 505–520
6. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC 2001), ACM (2001) 530–535
7. Goldberg, E., Novikov, Y.: BerkMin: A fast and robust SAT-solver. In: Proc. Design, Automation and Test in Europe Conference and Exposition (DATE 2002), Paris, France, IEEE Computer Society (2002) 131–149
8. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic 'phase transitions'. Nature **400** (1999) 133–137
9. Williams, R., Gomes, C., Selman, B.: Backdoors to typical case complexity. In: Intl. Joint. Conf. on Artificial Intelligence (IJCAI'03), Acapulco, Mexico (2003)
10. Rish, I., Dechter, R.: Resolution versus search: Two strategies for SAT. J. Automated Reasoning **24** (2000) 225–275
11. Park, T., Van Gelder, A.: Partitioning methods for satisfiability testing on large formulas. Information and Computation **162** (2000) 179–184
12. Gallo, G., Longo, G., Pallottino, S.: Directed hypergraphs and applications. Discrete Applied Mathematics **42** (1993) 177–201
13. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99). Number 1579 in LNCS, Springer-Verlag (1999)