Investigating Characteristics and Differences Between Easy and Hard SAT Instances (Extended Abstract)

Teofil Sidoruk[1]

Institute of Computer Science PAS, Warsaw
t.sidoruk@ipipan.waw.pl

Abstract. Following our comparison of the efficiency of SAT-solvers [19, 20], we analyse DIMACS input files previously generated for benchmarking purposes in an attempt to pinpoint some common characteristics for the CNF formulas that were relatively easier to process, i.e., were verified faster than comparable instances of the same size.

1 Introduction

Since the early 1970s, when Cooke first proved it to be NP-complete, the Boolean satisfiability problem, or SAT, has undergone a dramatic rise in importance. From the subject of purely theoretical research in the area of computational complexity, SAT-solving algorithms have become the cornerstone of a broad range of important practical applications that rely on their efficiency. They include, but are not limited to: verification [1, 2], (un)bounded model checking [5, 7, 13, 24, 25], planning [15], and composition of web services [18]. It is equally important to note that the theoretical aspects of SAT also remain the subject of keen scientific interest.

In our recent papers [19–21], we presented notable SAT-solvers, both state-of-the-art and historical, comparing their efficiency at several computational problems of varying complexity: from P-complete chess problems to EXPTIME-complete Towers of Hanoi puzzle. One obvious observation stemming from our comparison is that no single SAT-solver is superior to others in the sense that it always performs faster regardless of the input. In other words, solving SAT remains a considerable challenge: despite the incredible progress made, especially in the last fifteen years, the potential for further improvement is as large as ever.

The focus of this paper is not on SAT-solvers as such, but rather, on the input Boolean formulas themselves. Specifically, we will investigate CNF formulas that are comparatively easier or harder to verify compared to other generated instances of the same size, attempting to identify some common patterns in their properties. Given that this area that has not really been previously explored, this work is aiming to be an initial, small step rather than an exhaustive investigation.

The rest of this paper is organized as follows. The next section summarises existing work related to the subject. Section 3 shortly presents DIMACS, the standardised input format used by SAT-solvers, as well as details the generation and analysis of input CNF formulas. In Section 4 experimental results are compared and discussed. The final section contains conclusions.

2 Theoretical Overview and Related Work

In this section we discuss prior research into the difficulty of randomly generated instances of NP-complete problems.

Many well-known, classical computational problems, though NP-complete, are relatively easy to solve when it comes to typical instances [6]. The graph k-coloring problem, for example, was found to be solvable in logarithmic time in the vast majority of cases [23]. On the other hand, since their complexity was proven in Karp's seminal 1972 paper [14], we are bound to encounter hard instances eventually. This brings about the question of whether there is any pattern to be found in the distribution of difficulty in a set of randomly generated instances, which has been the subject of research since at least the early 1990s. In the rest of this paper, we will focus on the Boolean satisfiability problem (SAT), since it serves as the convenient 'common denominator' to which other hard problems are often translated.

It has been long observed that certain specific instances of SAT pose an unusually significant challenge to the the DPLL algorithm, contrary to perceived average difficulty. In [6] Cheeseman, Kanefsky and Taylor summarise classical NP-complete problems using 'order parameters'. For example, a set of instances of the Hamiltonian path problem can be ordered by the average connectivity of their respective graphs: the higher the connectivity, the higher the chance for a Hamiltonian path to exist. Furthermore, the authors show the existence of a phase transition at the boundary marked by some critical value of the order parameter, which separates two distinct regions of likely satisfiable and likely unsatisfiable instances, both of which are comparatively easy to verify. It is at the boundary that the hardest instances occur.

This phase transition is investigated further by Gent and Walsh in [11]. Their experimental results confirm the association of hardest instances of problems with the boundary, and that median problem difficulty generally follows the expected easy-hard-easy pattern. However, they also show that the distribution of difficulty is significantly more complex, and in particular note the presence of a region where instances can be extraordinarily difficult, sometimes orders of magnitude harder than those closest to the phase transition.

Gent and Walsh postulate the 'constraint gap' to cause such unexpectedly hard problems to occur in an otherwise satisfiable region. In the DPLL algorithm, neither unit propagation nor pure literal elimination ever branch out the search, leaving splitting (i.e., the choice of the branching literal) as the only critical point which can potentially result in an exponential blow-up in the number of explored assignments. This naturally leads to the conclusion that the harder the instance, the more the algorithm is forced to use the splitting rule compared to the other two. In other words, the hardest instances are 'constrained' in the sense they have just enough constraints to be unsatisfiable, but very few more (or even none), forcing DPLL to utilize heuristics-based branching and thus increasing verification time dramatically.

These results were further experimentally confirmed in other papers, including analyses for 3-SAT formulas by Larrabee and Tsuji [16] and by Crawford and Auton [8], with the latter work focusing on how the percentage of satisfiable instances changes as a function of the clause/variable ratio of the formula.

3 The Input Format, Formulas and Analysis

In this section we present the DIMACS input format commonly used by SAT-solvers, discuss the input files used for our analysis and the factors taken into account during the latter.

The renewed scientific interest in the Boolean satisfiability problem, and in particular the emergence of SAT Competitions in the early 2000s, resulted in the need of a single, unified input file format. DIMACS has become such a standard.

The format uses plain text to represent a Boolean formula in conjunctive normal form (CNF). Following an optional comment line and a header containing the number of clauses and literals in the formula, each subsequent line corresponds to a new clause. Variables are represented by subsequent natural numbers, with the minus sign denoting negation. Spaces separate literals in clauses, and zeroes signal end of clause. An example of a very simple CNF formula in the DIMACS format is shown below.

Listing 1.1. The input file corresponding to a simple formula $(x1 \vee \neg x3) \wedge (x2 \vee x3 \vee \neg x1)$

```
c Example DIMACS input
p cnf 3 2
1 -3 0
2 3 -1 0
```

The input files contain formulas resulting from the translations to SAT of several NP-complete problems, including classical graph problems (vertex coloring, vertex cover, Hamiltonian path) [14], as well as the extended string-to-string correction problem (ESCP). They were originally created as benchmarks for our previous work, i.e., a comparison of SAT-solvers [19, 20].

For the purposes of this analysis, we have identified and separated groups of the most and least difficult instances for each of the aforementioned problems, that is, the input files whose processing required the most and the least time, respectively. When calculating verification time, the average of individual solvers' processing times was considered. The solvers used were the same as in the aforementioned comparison: Lingeling and Plingeling [4], Glucose and Glucose-syrup [3], Clasp [10], Minisat [22], ManySAT [12] and Microsoft Z3 [9]. However, zChaff [17] was excluded due to its age and inability to process many instances in reasonable time, which would have considerably skewed the average.

The DIMACS files in both groups were subsequently analysed and compared w.r.t. factors such as total number of literals and clauses, average and maximum clause length, percentage of negative literals and percentage of Horn clauses.

4 Results

In this section we discuss and compare the results of our analysis.

In Tab. 1, several characteristics are compared between the groups of easiest and hardest generated instances of vertex k-colouring and vertex k-cover. For the former, parameters of $n = 100$ (graph size) and $k = 10$ (number of colours) were set. For the latter, graphs of size $n = 50$ were generated, with the vertex cover size at $k = 30$.

	Vertex k-colouring		Vertex k-cover	
	Easier instances	Harder instances	Easier instances	Harder instances
Avg running time	0.018 s	364.759 s	0.108 s	38.941 s
Avg number of clauses	24507	36007	36774	36994
Avg number of literals	49813	72813	74967	88127
Avg clause size	2.033	2.022	2.039	2.382
Longest clause	10	10	60	60
Negative literals	97.96%	98.63%	98.04%	83.41%
Horn clauses	0%	0%	99.03%	99.34%

Table 1. Comparison of characteristics between easier and harder instances of the vertex k-colouring and k-cover problems.

	Hamiltonian path		String correction	
	Easier instances	Harder instances	Easier instances	Harder instances
Avg running time	4.810 s	34.101 s	0.538 s	42.968 s
Avg number of clauses	8000200	8000200	49808	49808
Avg number of literals	20135132	17115771	318476	318476
Avg clause size	2.517	2.139	6.394	6.394
Longest clause	200	200	66	66
Negative literals	80.66%	93.25%	19.67%	19.67%
Horn clauses	0%	0%	0%	0.01%

Table 2. Comparison of characteristics between easier and harder instances of the Hamiltonian path problem and ESCP.

It can be observed that the characteristics of hard instances depend primarily on the computational problem and its specific translation to SAT. For instance, in the vertex k-colouring problem, they have up to 50% more literals and clauses. This in turn can be attributed to the randomly generated input graphs for these instances having significantly more edges, and as such requiring more constraints in the form of clauses. Notably, despite the overwhelming majority of literals in both groups of instances being negative, there are no Horn clauses, again due to the specifics of the SAT encoding.

On the other hand, for vertex k-cover the average number of clauses is roughly the same in easy and hard instances. However, the number of literals, and thus average clause length, is generally higher (up to 15%) in the latter group. Furthermore, an even more noteworthy difference is in the percentage of negative literals, which is also around 15% lower, suggesting that the extra literals in hard instances are positive.

In the same way, Tab. 2 compares instances of the Hamiltonian path problem and the extended string-to-string correction problem (ESCP). For the former, generated graphs were of size $n = 200$, whereas the parameters for ESCP were set as $n = 20$ (length of input strings), $k = 15$ (maximum number of operations) and $l = 5$ (alphabet size).

In the case of the Hamiltonian path problem, the harder instances actually have less literals and thus, on average, shorter clauses. This, too, is consistent with the nature of the problem in question: fewer edges (and thus shorter conditional clauses in the resultant formula) make for a graph that is harder to find a Hamiltonian path, while the most trivial satisfiable instance is actually one in which all possible edges exist.

Finally, in the comparison of ESCP instances, all analysed characteristics are virtually identical, further emphasising the lack of any clear pattern behind the relative difficulty of specific instances of SAT.

It is important to note that our analysis is not yet another attempt to back up the the findings previously described in Section 2, i.e., the existence of a phase transition and a 'constraint gap' at the boundary between regions of expected (un)satisfiability. Instead of considering the distribution of difficulty across some order parameter, we took into account benchmarks generated using the same settings, i.e., the same order parameter, in an attempt to pinpoint patterns related to the composition of the formulas themselves. However, it clearly appears that the differences are related to the specific characteristics of computational problems translated to SAT.

5 Conclusions

We have analysed Boolean formulas in CNF, representing translations of well-known NP-complete problems to SAT. The input files were grouped depending on their average processing time by SAT-solvers, and compared on several factors, including average clause length and percentage of Horn clauses, between the easiest and most difficult instances.

There do not appear to be easily noticeable global characteristics of CNF formulas representing harder instances of NP-complete problems. Depending on the specific problem and its translation to SAT, the formulas whose processing takes longer can, for instance, have longer clauses, or conversely, more clauses of similar average length to that in the 'easier' group. Similarly, the percentage of negative literals or Horn clauses

are also dependent on the NP-complete problem translated to SAT, and not some pattern prevalent across all comparatively easier or harder instances.

These observations seem in line with SAT being NP-complete, and as such, a difficult computational problem. Just as there is not a single SAT-solver always offering superior performance, no single factor contributes to a particular instance of SAT being comparatively easier or harder to verify than others of same size. This was most evident in the ESCP comparison: characteristics of both groups of instances were nearly identical, clearly showing that we cannot expect easy answers when it comes to hard computational problems. At least, not yet.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In *Proc. of the 6th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 411–425. Springer-Verlag, 2000.
2. A. Armando and L. Compagna. An optimized intruder model for SAT-based model-checking of security protocols. In *Electronic Notes in Theoretical Computer Science*, volume 125, pages 91–108. Elsevier Science Publishers, March 2005.
3. G. Audemard and L. Simon. Glucose and syrup in the SAT race 2015. *SAT Race*, 2015.
4. A. Biere. Lingeling and friends entering the SAT challenge 2012. *Proceedings of SAT Challenge*, pages 33–34, 2012.
5. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of the ACM/IEEE Design Automation Conference (DAC)*, pages 317–320, 1999.
6. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, pages 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.
7. E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
8. J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1):31 – 57, 1996. Frontiers in Problem Solving: Phase Transitions and Complexity.
9. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer-Verlag, 2008.
10. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp : A conflict-driven answer set solver. In *LPNMR*, 2007.
11. I. P. Gent and T. Walsh. The hardest random SAT problems. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence*, pages 355–366, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
12. Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: a Parallel SAT Solver. *JSAT*, 6(4):245–262, 2009.
13. M. Kacprzak and W. Penczek. A SAT-based approach to unbounded model checking for alternating-time temporal epistemic logic. *Synthese*, 142:203–227, 2004.
14. R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
15. H. Kautz and B. Selman. Planning as satisfiability. In *ECAI 92: Proceedings of the 10th European conference on Artificial intelligence*, pages 359–363, 1992.

16. T. Larrabee and Y. Tsuji. Evidence for a satisfiability threshold for random 3CNF formulas. Technical report, 1993.
17. Y. Mahajan, Z. Fu, and S. Malik. Zchaff2004: An efficient sat solver. In *Int. Conf. on Theory and Applications of Satisfiability Testing*, pages 360–375. Springer, 2004.
18. A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, and A. Zbrzezny. Towards automatic composition of web services: SAT-based concretisation of abstract scenarios. *Fundam. Inform.*, 120(2):181–203, 2012.
19. A. Niewiadomski, W. Penczek, and T. Sidoruk. Comparing Efficiency of Modern SAT-solvers for Selected Problems in P, NP, PSPACE, and EXPTIME. In *Proceedings of the 26th International Workshop on Concurrency, Specification and Programming, Warsaw, Poland, September 25-27, 2017*, pages 1–12, 2017.
20. A. Niewiadomski, P. Switalski, W. Penczek, and T. Sidoruk. Applying Modern SAT-solvers to Solving Hard Problems. *Fundamenta Informaticae (submitted)*, 2018.
21. A. Niewiadomski, P. Switalski, W. Penczek, and T. Sidoruk. SMT-solvers in action: encoding and solving selected problems in NP and EXPTIME. *Scientific Annals of Computer Science (submitted)*, 2018.
22. N. Sorensson and N. Een. Minisat v1. 13 - a SAT solver with conflict-clause minimization. *SAT*, 2005(53):1–2, 2005.
23. J. S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9(1):63 – 82, 1988.
24. B. Woźna, A. Zbrzezny, and W. Penczek. Checking reachability properties for Timed Automata via SAT. *Fundamenta Informaticae*, 55(2):223–241, 2003.
25. B. Woźna-Szczesniak. SAT-based bounded model checking for weighted deontic interpreted systems. *Fundam. Inform.*, 143(1-2):173–205, 2016.