SATO: an Efficient Propositional Prover

Hantao Zhang[*]

Department of Computer Science
The University of Iowa
Iowa City, IA 52242-1419, USA
hzhang@cs.uiowa.edu

SATO (Satisfiability Testing Optimized) is a propositional prover based on the Davis-Putnam method [3], which is is one of the major practical methods for the satisfiability (SAT) problem of propositional logic. The first report of SATO appeared in [12]. Since then, we constantly add new techniques into SATO to make it more efficient [14, 13].

One of the major motivations to develop SATO was to attack open Latin square problems. While SATO works well on Latin square problems, its previous versions did not work well on many classes of the SAT problem. In the fall of 1996, we made an effort to improve SATO so that it works well on a large set of the SAT problem. In the following, we discuss briefly two techniques that we found effective to improve SATO performance. One is about splitting rules; the other is about conflict analysis. While these two techniques are known in the community, the real challenge is how to integrate these techniques without weakening each other. We are happy to report here that the two techniques integrated very well with the techniques previously implemented in SATO. In the following discussions, we assume that the reader is familiar with propositional logic and the Davis-Putnam method [3].

One important place where heuristics may be inserted in the Davis-Putnam method is in the choice of a literal for splitting. It is well-known that different splitting rules make the performance of the Davis-Putnam algorithm different by a magnitude of several orders. While SATO provides several popular splitting rules, each rule works well only for a particular class of SAT instances.

For instance, in our study of quasigroup problems, one rule seems better than the others: choose one literal in one of the shortest positive clauses (a positive clause is a clause where all the literals are positive). On the other hand, a proved effective splitting rule is to choose a variable x such that the value $f_2(x) * f_2(\neg x)$ is maximal, where $f_2(L)$ is one plus the number of occurrences of literal L in binary clauses [2, 5].

We tried to combine the above two rules into one as follows: Let $0 < a \leq 1$ and n be the number of shortest non-Horn clauses in the current set. At first, we collect all the variable names appearing in the first $\lceil a * n \rceil$ shortest positive clauses. Then we choose x in this pool of the variables with the maximal value $f_2(x) * f_2(\neg x)$. We found that this mixed rule worked quite well if a is the percentage of non-Horn clauses in the input multiplied by 5. That is, the splitting rule adjusts by itself for various input clauses.

[*] Supported in part by NSF under grants CCR-9504205 and CCR-9357851.

Intelligent backjumping has been extensively studied in the search procedures for solving constraint satisfaction problems [10]. Since the SAT problem is a special case of constraint satisfaction, many researchers have applied this technique in the Davis-Putnam method [7].

The basic idea is very simple: A literal is assigned to `true` in the Davis-Putnam method either by the splitting rule (active) or by unit propagation (passive). When an empty clause is found, backtracking is needed. At this point, we may collect all the active literals which played a role in the making of the empty clause. If the current active literal does not belong to this set, we do not need to try the second truth value of this literal. That is so-called "intelligent backjumping".

In order to avoid the collection of the same set of literals at a later stage of the search, we may save the disjunction of the negations of these collected literals as a new clause in the system. Silva and Sakllah used the same idea in the Davis-Putnam method and reported very good experimental results [9]. We also implemented this idea as well as the self-adjusting splitting rule in the latest version of SATO, i.e., SATO 3.0.

This technique of creating new clauses does not always improve the performance. In fact, for certain SAT instances, the performance becomes much worse because the prover spends much time and memory storing these new clauses. Thus, an important parameter that greatly affects the performance is the maximal length of the newly created clauses allowed to be saved.

In 1993-1994, the participants of The Second DIMACS Implementation Challenge collected a large set of SAT instances from different application areas. This set of SAT instances is called *the DIMACS benchmarks*, available on the internet from [6].

In Table 1, we present experimental results of SATO 3.0 on the DIMACS benchmarks, in comparison with other state-of-the-art and publicly available SAT provers, including GRASP [9], POSIT [5], C-SAT [4], H2R [8], NTAB [2] (the latest version of TABLEAU), TEGUS [11], DPL [1].[2] The CPU times for these provers are obtained from [9]. All of the times were scaled to the equivalent CPU times on a SUN SPARC 5/85 machine, using the test program provided at DIMACS [6] for comparing different machine architectures.[3]

In the first column of the table, the class names of SAT instances are given. The number under each class name is the number of the instances in that class. Three classes of the problems in the DIMACS benchmarks are not included in the table: `par32` (10 instances), `f` (three instances), and `g` (four instances). The reason is that none of the provers listed in the table can solve any instance in these three classes in less than 10,000 seconds of CPU time. Of each entry in the

[2] Incomplete propositional provers like GSAT are excluded here because many instances are unsatisfiable.

[3] The execution times of DFMAX on the SUN SPARC 5/85 are 0.03, 0.68, 6.12, 39.05, and 149.92 (seconds), respectively, for r100, ..., r500. SATO3 was compiled in gcc with the option -O3 on a SGI Onyx machine. The run times of DFMAX on this machine are: 0.01, 0.36, 3.06, 18.85, and 72.0.

Prob.	GRASP	POSIT	H2R	C-SAT	NTAB	TEGUS	DPL	SATO2	SATO3
`aim-50`	0.4	0.4	2.3	10,002	24.3	2.2	10.7	12.7	0.02
24	24	24	24	23	24	24	24	24	24
`aim-100`	1.8	1,290	21,571	5.1	39,569	107.9	58,510	60,390	0.4
24	24	24	23	24	18	24	21	20	24
`aim-200`	10.8	117,991	150,004	50,043	69,410	14,059	156,196	150,095	1.1
24	24	13	9	19	11	23	9	9	24
`bf`	7.2	20,037	10,200	30,509	27,900	26,654	40,000	35,695	2.9
4	4	2	3	1	2	2	0	1	4
`dubois`	34.4	77,189	73,729	79,620	47,952	90,333	96,977	71,528	1.5
13	13	7	7	7	5	5	5	7	13
`hanoi`	14,480	10,117	10,733	15,533	15,840	11,641	20,000	20,000	10,001
2	1	1	1	1	1	1	0	0	1
`hole`	12,704	937.9	11,182	858.9	1,244	21,301	11,404	362.2	841.7
5	4	5	4	5	5	3	4	5	5
`ii8`	23.4	2.3	30,005	16,966	11,411	11.8	84,189	0.4	1.1
14	14	14	11	13	13	14	7	14	14
`ii16`	10,311	10,120	75,940	50,489	10,126	269.6	83,933	85,522	5.3
10	9	9	3	5	6	10	2	7	10
`ii32`	7.0	650.1	36,029	170,000	697.0	1,231	21,520	10,004	11.8
17	17	17	14	0	17	17	15	16	17
`jnh`	21.3	0.8	5.8	7.6	10.9	6,055	40.0	11.0	2.1
50	50	50	50	50	50	50	50	50	50
`par8`	0.4	0.1	0.6	70,019	0.7	1.5	0.8	0.2	0.2
10	10	10	10	3	10	10	10	10	10
`par16`	9,844	72.1	264.8	70,809	591.5	9,983	11,741	10,447	607
10	10	10	10	3	10	10	10	10	10
`pret`	18.2	40,691	40,342	41,201	80,000	42,579	41,429	40,430	3.0
8	8	4	4	4	0	4	4	4	8
`ssa`	6.5	85.3	20,006	14,903	20,024	20,230	80,000	30,092	4.0
8	8	8	6	7	6	6	0	5	8

Table 1. Experimental Results on the DIMACS benchmarks

table, the first number is the cumulated CPU time spent by a prover for that class of the problems (if a prover cannot solve an instance under 10,000 seconds, the assumed CPU time is 10,000 seconds); the second number is the number of the instances solved by the prover in that class.

SATO2 denotes SATO 2.2 and SATO3 denotes SATO 3.0. The former is an older version of SATO that does not use the intelligent backjumping and the self-adjusting splitting rule. For this experiment with SATO 3.0, the maximal length of the newly created clauses allowed to be saved is 20. It is apparent that SATO 3.0 is significantly faster than SATO 2.2 on the DIMACS benchmarks (except two classes, `ii8` and `hole`). In fact, comparing with all the provers listed in the table, the performance of SATO 3.0 is either the best or the second best for every class of the problems, except `par16`.

The code of SATO 3.0 is available from World Wide Web at

http://www.cs.uiowa.edu/~hzhang/sato.html

References

1. Barth, P., A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization, Technical Report MPI-I-95-2-003, Max-Plank-Institut fur Informatik, 1995.
2. Crawford, J., Auton, L., (1993) Experimental results on the cross-over point in satisfiability problems. In *Proc. of the 11th National Conference on Artificial Intelligence* (AAAI-93), pp. 22-28.
3. Davis, M., G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery 5,* 7 (July 1962), 394–397.
4. Dubois, O, Andre, P., Boufkhan, Y., Carlier, J., SAT versus UNSAT, see [6].
5. (1995) Freeman, J.W., Improvements to propositional satisfiability search algorithms. Ph.D. Dissertation, Dept. of Computer Science, University of Pennsylvania.
6. Johnson, D.S., Trick, M.A., (eds.) The second DIMACS implementation challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (see http://dimacs.rutgers.edu/challenges/)
7. (1980) McAllester, D.A., An outlook on truth maintenance, AI Memo 551, MIT AI laboratory.
8. Pretolani, D., Efficiency and stability of hypergraph SAT algorithms, see [6].
9. Silva, J.P.M., Sakallah, K.A., Conflict analysis in search algorithms for propositional satisfiability. Technical Reports, Cadence European Laboraties, ALGOS, INESC, Lisboa, Portugal, May 1996.
10. Stallman, R.M., Sussman, G.J., (1977) Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, vol. 9, 135-196
11. Stephan, P.R., Brayton, R.K., Sangiovanni-Binventelli, Combinational test generation using satisfiability, Memo no. UCS/ERL M92/112, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, Oct. 1992.
12. Zhang, H., SATO: A decision procedure for propositional logic. Association for Automated Reasoning Newsletter, **22**, 1–3, March 1993.
13. Zhang, H., Bonacina, M.P., Hsiang, H.: (1996) PSATO: a distributed propositional prover and its application to quasigroup problems. To appear in Journal of Symbolic Computation.
14. Zhang, H., Stickel, M. (1994) Implementing the Davis-Putnam algorithm by tries. Technical Report, Dept. of Computer Science, The University of Iowa.

This article was processed using the LaTeX macro package with LLNCS style