SAT with Partial Clauses and Back-Leaps

Slawomir Pilarski and Gracia Hu
Synopsys, Inc.
2025 NW Cornelius Pass Rd.
Hillsboro, OR 97124

Abstract

This paper presents four new powerful SAT optimization techniques: *partial clauses*, *back-leaps*, *immediate implications*, and *local decisions*. These optimization techniques can be combined with SAT mechanisms used in Chaff, SATO, and GRASP to develop a new solver that significantly outperforms its predecessors on a large set of benchmarks. Performance improvements for standard benchmark groups vary from 1.5x to 60x. Performance improvements measured using VLIW microprocessor benchmarks amount to 3.31x.

1 Introduction

Boolean Satisfiability (SAT) solvers have been successfully applied to solve various problems in EDA [8]. They are useful in formal verification for combinational and sequential equivalence checking [5], property checking, and microprocessor verification [16]. They are known to be effective in automatic test pattern generation (ATPG) [6, 7, 14], which includes delay fault testing [1] and redundancy removal. SAT solvers were used in logic synthesis [2], timing analysis, and FPGA routing [12, 11].

The wide range of successful SAT applications in EDA is due to the tremendous progress in the performance of SAT solvers [9, 17]. A recently developed solver, Chaff, demonstrated further dramatic performance gains [10, 16, 18]. The speed of Chaff is due to very efficient implementation of all mechanisms used in a standard backtracking search algorithm.

This paper introduces four new powerful SAT optimization techniques: *partial clauses*, *back-leaps*, *immediate implications*, and *local decisions*. It demonstrates that the new optimization techniques can be successfully implemented together with SAT mechanisms used in most efficient SAT solvers. The resulting new solver significantly outperforms its predecessors on a large set of benchmarks.

The paper is organized as follows. Improvements to boolean constraint propagation (BCP) are introduced in Section 2. New SAT decision optimization techniques are presented in Sections 3.1 and 3.2. Benchmark results are given in Section 4. Concluding remarks are presented in Section 5.

Table 1: Average BCP time per implication.

benchmark	no optim.	imm. impl.	speedup
5pipe	800 cycles	599 cycles	1.33x
bug001	686 cycles	606 cycles	1.13x

2 Improvements to BCP

Efficient boolean constraint propagation based on watched literals [17, 10] can be further improved by using *immediate implications* and *partial clauses*. Both concepts are presented below.

2.1 Immediate Implications

EDA-related CNF formulae are usually derived from logic circuits. For example, a simple AND gate, $x = AND(y, z)$, can be expressed as $(\overline{x}+y)(\overline{x}+z)(x+\overline{y}+\overline{z})$. It is easy to note that CNF for AND, NAND, OR and NOR gates are very similar, and they are dominated by two-literal clauses. In a typical circuit-derived CNF formula, 55%-85% of all clauses have two literals. Now, note that a 0-assignment to a watched literal in a two-literal clause is an *immediate implication*, and there is no need to search the clause for non-watched literals, as it happens in Chaff or SATO. The notion of immediate implications can be used to speed up BCP. In the case of an already highly optimized BCP, immediate implications can noticeably improve performance [13]. Table 1 presents examples of average time in machine cycles spent on BCP per implication with and without immediate implications. Note that various benchmarks may have very different speedups – not always as significant.

We also found out that SAT performance can be further improved if BCP processes immediate implications first.

Table 2: Average BCP time per implication.

benchmark	no optim.	optim.	speedup
5pipe	800 cycles	330 cycles	2.42x
bug001	686 cycles	369 cycles	1.86x

This, however, appears to be important for conflict clause selection, not for BCP speed.

2.2 Partial Clauses

Conflict clauses are usually much longer than the original benchmark clauses. The average conflict clause size often exceeds 200 or even 300 literals. Since most literals in an undeleted conflict clause are assigned to logic 0, a significant fraction of BCP time is spent on finding the next watched literal. This overhead could be reduced if, during BCP, unsatisfied clauses are temporarily replaced with clauses consisting of only unassigned literals. Such a transformation would not affect SAT correctness since 0-assigned literals do not affect BCP, except for slowing it down. Our observation leads to the concept of a *partial clause*. A partial clause is created by elimination of most 0-assigned literals in its parent clause. It can be used instead of its parent clause only when all eliminated literals are assigned to logic 0. For example clause

$(v_1{}^{=0} + v_2{}^{=0} + v_3{}^{=0} + v_4{}^{=0} + v_5{}^{=?} + v_6{}^{=?} + v_7{}^{=?})$,

where literals v_1, v_2, v_3, v_4 are assigned to logic 0 and literals v_5, v_6, v_7 are unassigned, can be replaced with partial clause

$(v_4{}^{=0} + v_5{}^{=?} + v_6{}^{=?} + v_7{}^{=?})$.

In our implementation, validity of a partial clause is linked to the highest decision level associated with eliminated literals. Partial clause inherits watched literals from its parent clause. Only conflict clauses are replaced with partial clauses.

Partial clauses together with immediate implications provide effective BCP optimization. Table 2 presents examples of average time in machine cycles spent on BCP per implication; column "no optim." shows data for BCP without optimizations; column "optim." shows data for BCP with immediate implications and partial clauses. Note that various benchmarks may have very different speedups – not always as significant. In general, the speedup due to partial clauses depends on the average size of a conflict clause; it also depends on the fraction of conflict clauses in the database, not to mention quality of partial clause implementation.

3 Improvements to Decisions

Dynamic decision making mechanism used in Chaff can be improved by using *local decisions* and *back-leaps*. Both concepts are presented below.

3.1 Local Decisions

In general, as in many other SAT solvers, the decision process used in Chaff selects a variable regardless of its relation to currently assigned variables. This means that non-chronological backtracking often invalidates many decisions and BCP computations unrelated to the conflict. Such an observation suggests that a significant performance gain is possible if the decision process could be focused on variables related to currently assigned variables.

In EDA formulae conflicts often express a correlation between variables that are close in terms of circuit connectivity. This suggests that the decision process should be biased towards unassigned variables that share clauses with already assigned variables. Relatively strong correlation between variables distant in terms of circuit connectivity can also be observed. Because of this, too much focus on circuit structure may lead to suboptimal decisions and conflict clauses that prune the search space relatively slowly.

Our decision strategy has two components:

- *Local:* select the "best" variable that shares a clause with already assigned variables; and
- *Global:* select globally "best" variable.

The mechanism of ranking variables in both groups is similar, though not identical, to that used in Chaff. We use two priority queues one for local and one for global decisions. We put less emphasis on literals in two-literal clauses – they carry smaller weight than other literals. About 80% of decisions select the locally "best" variable.

Our structure-oriented decisions appear to be especially effective for hard-to-satisfy and unsatisfiable formulae. Table 3 reports the impact of local decisions on SAT run time measured for CMU benchmarks. Note that our implementation also used immediate implications, but did not use partial clauses.

3.2 Back-Leaps

Standard non-chronological backtracking is a powerful mechanism used to prune the search space, but most backtracks are are still chronological (see Table 3 in [9]). This observation suggests that a SAT solver would benefit from a mechanism that guides the decision sequence out of chronological backtrack regions. A crude version of such

Table 3: CMU benchmark [15] results (in seconds).

benchmark	#tests	zChaff	local d.	speedup
sss 1.0	48	99	**33**	2.96x
sss 1.0a	8	37	**19**	1.86x
sss-sat 1.0	100	672	**358**	1.88x
fvp-unsat-1.0	4	1480	**839**	1.76x
fvp-unsat-2.0	22	218268	**55081**	3.96x
vliw-sat 1.0	100	13626	**9237**	1.48x

a mechanism is provided by restarts. A more sophisticated version of such a mechanism would monitor a number of most recent backtracks; if the majority of them are chronological, cancellation of some recent decisions, *a back-leap*, could be forced. Unlike backtracking, back-leaping does not flip the value of the variable truth assignment. This means that introduction of back-leaping does not compromise SAT solver completeness.

In our implementation we monitor ten most recent backtracks; if the total number of backtracked levels is smaller than a threshold value, we consider a back-leap to the second most recent decision level found in the the conflict clause; a back-leap takes place only if we can back-leap at least five decisions. For example, conflict clause $(v_1{}^{=0,level=3} + v_2{}^{=0,level=7} + v_3{}^{=0,level=15} + v_4{}^{=0,level=22} + v_5{}^{=0,level=23} + v_6{}^{=0,level=33} + v_7{}^{=0,level=45})$ may trigger back-leaping to level 33 (since $45 - 33 > 5$).

Back-leaping seems to be especially effective for relatively hard-to-satisfy formulae, but not very effective for some unsatisfiable formulae.

A combination of back-leaping, local decisions, immediate implications and partial clauses translate into significant performance gains.

4 Experimental Results

This section presents performance results for a SAT solver that combines effective SAT techniques used in GRASP, SATO and Chaff with the powerful SAT optimizations we introduced in Sections 2 and 3. Table 4 summarizes results for the DIMACs benchmark suite [4]. Each row presents benchmark name, number of tests in the benchmark, run time (in seconds) for GRASP, SATO, and Chaff, run time for the new solver, and the speed up factor with respect to Chaff. Table 5 summarizes results for the CMU Benchmark Suite [15]. Both sets of benchmarks were used in [10] to illustrate performance gains offered by Chaff. We use the same benchmarks to demonstrate performance gains due to our SAT optimizations.

Note that all experiments reported in Tables 4 and 5 were run on the same 4CPU/400MHz/4GB UltraSPARC-4 machine. We used original unmodified GRASP, SATO, and zChaff packages with the following settings: GRASP(+T100 +B10000000 +C10000000 +S10000 +g20 +rt4 +dDLIS), SATO(default), and zChaff(default). Best results are highlighted. A dash means that at least one test was timed out.

5 Conclusions

This paper demonstrates that the new four SAT optimization techniques introduced in Sections 2 can be successfully implemented together with efficient SAT mechanisms used in Chaff, SATO, and GRASP. The resulting new SAT solver is optimized for EDA applications. Benchmark results show performance improvements with respect to Chaff ranging from 1.5x to 60x. It appears that a revised variable scoring mechanism (e.g., the one used in [3]) could amplify already significant performance gains.

References

[1] C.-A. Chen and S. K. Gupta. A satisfiability-based test generator for path delay faults in combinational circuits. In *Proc.Design Automation Conference*, pages 209–214, June 1996.

[2] L. Entrena and K.-T. Cheng. Sequential logic optimization by redundancy addition and removal. In *Proc. International Conference on Computer-Aided Design*, pages 310–315, November 1993.

[3] Evgueni Goldberg and Yakov Novikov. Berkmin: a fast and robust sat-solver. In *Proc. Design Automation and Test in Europe*, pages 142–149, March 2002.

[4] D. S. Johnson and M. A. Trick eds. Second DIMACS implementation challenge. In *DIMACS benchmarks available at ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks*, 1993.

[5] Andreas Kuehlmann, Malay K. Ganai, and Viresh Paruthi. Circuit-based boolean reasoning. In *Proc. ACM/IEEE Design Automation Conference*, pages 232–237, June 2001.

[6] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Trans. CAD*, 11(1):4–15, January 1992.

[7] J. Marques-Silva and K. A. Sakallah. Robust search algorithms for test pattern generation. In *Proc. of the Fault-Tolerant Computing Symposium*, pages 152–161, June 1997.

Table 4: DIMACs benchmark results (in seconds)[4].

benchmark	# tests	GRASP	SATO	zChaff	new Solver	speed up w.r.t. zChaff
ii16	10	-	2.18	67.71	**1.62**	41.80x
ii32	17	4.47	2.47	2.75	**1.06**	2.59x
ii8	14	2.03	0.48	0.53	**0.04**	13.25x
aim200	24	5.81	0.77	1.11	**0.37**	3.00x
aim100	24	0.70	0.20	0.20	**0.04**	5.0x
pret	8	4.34	**0.07**	1.13	0.14	8.07x
par8	10	0.25	0.07	0.02	**0.01**	2.00x
ssa	8	3.30	3.78	0.28	**0.27**	1.04x
jnh	50	5.69	1.04	0.72	**0.47**	1.53x
dubois	13	0.35	0.17	0.14	**0.05**	2.08x
hole	5	-	142.73	**42.76**	48.61	0.88x
par16	10	-	-	31.77	**28.51**	1.11x
hanoi 5	1	-	-	76705	**1299**	59.05x

Table 5: CMU benchmark results (in seconds)[15].

benchmark	# tests	zChaff	new Solver	speedup
sss 1.0	48	99.14	**40.39**	2.45x
sss 1.0a	8	36.72	**8.56**	4.29x
sss-sat 1.0	100	671.75	**278.76**	2.41x
fvp-unsat-1.0	4	1480.24	**216.97**	6.82x
fvp-unsat-2.0	22	218268	**92987**	2.35x
vliw-sat 1.0	100	13626	**4115**	3.31x

[8] J. Marques-Silva and K. A. Sakallah. Boolean satisfiability in electronic design automation. In *Proc. ACM/IEEE Design Automation Conference*, pages 675–680, June 2000.

[9] Joao P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comput*, 48(5):506–520, May 1999.

[10] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc.Design Automation Conference*, pages 530–535, June 2001.

[11] G.-J. Nam, K.A. Sakallah, and R. Rutenbar. Satisfiability based FPGA routing. In *Proc. of the International Conference on VLSI Design*, pages 574–577, January 1999.

[12] G.-J. Nam, K.A. Sakallah, and R. Rutenbar. Satisfiability-based layout revisited: Detailed routing of complex FPGAs via search-based boolean SAT. In *Proc. of the International Symposium on Field- Programmable Gate Arrays*, pages 167–175, February 1999.

[13] Slawomir Pilarski and Gracia Hu. Speeding up SAT for EDA. In *Proc. Design Automation and Test in Europe*, page 1081, March 2002.

[14] P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. CAD*, 15(9):1167–1176, September 1996.

[15] M. N. Velev. CMU benchmark suite. In *benchmarks available at htpp://www.ece.cmu.edu/m̃velev*.

[16] Miroslav N. Velev and Randal E. Bryant. Effective use of boolean satisfiability procedure in the formal verification of superscalar and VLIW microprocessors. In *Proc. Design Automation Conference*, pages 226–231, June 2001.

[17] H. Zhang. SATO: An efficient propositional prover. In *Proc. of the International Conference on Automated Deduction*, pages 272–275, July 1997.

[18] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proc. International Conference on Computer-Aided Design*, pages 279–285, November 2001.