Multiple Preprocessing for Systematic SAT Solvers

Anbulagan[1], John Slaney[1,2]
[1]Logic and Computation Program, National ICT Australia Ltd.
[2]Computer Sciences Laboratory, Australian National University
{anbulagan, john.slaney}@nicta.com.au

Abstract

High-performance SAT solvers based on systematic search generally use either conflict driven clause learning (CDCL) or lookahead techniques to gain efficiency. Both styles of reasoning can gain from a preprocessing phase in which some form of deduction is used to simplify the problem. In this paper we undertake an empirical examination of the effects of several recently proposed preprocessors on both CDCL and lookahead-based SAT solvers. One finding is that the use of multiple preprocessors one after the other can be much more effective than using any one of them alone, but that the order in which they are applied is significant. We intend our results to be particularly useful to those implementing new preprocessors and solvers.

1 Introduction

In the last decade, the propositional satisfiability (SAT) has become one of the most interesting research problems within artificial intelligence (AI). This tendency can be seen through the development of a number of powerful SAT solvers, based on either systematic search or stochastic local search (SLS), for solving various hard combinatorial search problems such as automatic deduction, hardware and software verification, planning, scheduling, and FPGA routing.

The power of contemporary systematic SAT solvers derives not only from the underlying Davis-Putnam-Logemann-Loveland (DPLL) algorithm but also from enhancements aimed at increasing the amount of unit propagation, improving the choices of variables for splitting or making backtracking more intelligent. Two of the most important such enhancements are conflict driven clause learning (CDCL), made practicable on a large scale by the watched literal technique, and one-step lookahead. These two tend to exclude each other: the most successful solvers generally incorporate one or the other but not both. The benefits they bring are rather different too, as is clear from the results of recent SAT competitions. For problems in the "industrial" category, CDCL, as implemented in MiniSat [ES03, SE05], siege [Rya04] and zChaff [MMZ+01, ZMMM01] is currently the method of choice. On random problems, however, lookahead-based solvers such as Dew_Satz [AS05], Kcnfs [DD01] and March_dl [HvM06] perform better.

Lookahead, of course, is expensive at every choice node, while clause learning is expensive only at backtrack points. Since half of the nodes (plus one) in any binary tree are leaves, this difference is significant for lookahead-based solvers which process

nodes relatively slowly and gain, if at all, by reducing the search tree size. Looking ahead is an investment in at-node processing which can pay off only if it results in more informed choices with an impact on the total number of nodes visited. Where learnt clauses prune the tree at least as effectively as complex choice heuristics, CDCL must win. This seems to be the case in many classes of highly structured problems such as the "industrial" ones in the SAT competitions. We have no clearer definition than anyone else of "structure", but are interested to find ways in which lookahead-based solvers might detect and exploit it as well as the clause learners do.

A noteworthy feature of many recent systems is a preprocessing phase, often using inference by some variant of resolution, to transform problems prior to the search. One suggestion we wish to make and explore is that such transformations may help lookahead-based solvers to discover useful structure. That is, much of the reasoning done by nogood inference might be done cheaply "up-front", provided that the subsequent variable choice heuristics are good enough to exploit it. In what follows, we are therefore concerned mainly with the effects of preprocessing, including those of using multiple preprocessors in series, on the performance of a lookahead-based solver Dew_Satz on problems where it does poorly in comparison with a clause learning solver (MiniSat). We also include some results and remarks on benefits to be gained in the opposite direction, where MiniSat is helped by preprocessing to attack problems to which Dew_Satz is more suited.

In this paper we propose a multiple preprocessing technique to boost the performance of systematic SAT solvers. The motivation for applying multiple preprocessors prior to the systematic search process is clear: each preprocessor uses a different strategy in objective to simplify clause sets derived from real-world problems that exhibit a great deal structure such as symmetries, variable dependencies, clustering, and the like. Our initial observation showed that each strategy works well for simplifying the structure of some problems, at most of the time, from hard to easy. When a problem exhibits different kinds of structure, then a single preprocessor has difficulty to simplify the structures. In this case, we need to run multiple preprocessors one after the other.

We report performance statistics for the two solvers, Dew_Satz and MiniSat, with and without combinations of five (and for one problem set, six) of the best contemporary SAT preprocessors when solving parity, planning, bounded model checking and FPGA routing benchmark problems from SATLIB and the recent SAT competitions. One finding is that the use of multiple preprocessors one after the other can be much more effective than using any one of them alone, but that the order in which they are applied is significant. We intend our results to be particularly useful to those implementing new preprocessors and solvers.

The rest of the paper is organized as follows: section 2 addresses the related work. In sections 3 and 4, we briefly describe the preprocessors and solvers examined in our study. The main part of the paper consists of experimental results, and we conclude with a few remarks and suggestions.

2 Related Work

Resolution-based SAT preprocessors for CNF formula simplification have a dramatic impact on the performance of even the most efficient SAT solvers on many benchmark problems [LMS01]. The simplest preprocessor consists of just computing length-bounded resolvents and deleting duplicate and subsumed clauses, as well as tautologies and any duplicate literals in a clause.

There are two most directly related works. The first one is that of Anbulagan *et al.* [APSS06] which examined the integration of five resolution-based preprocessors alone or the combination of them with stochastic local search (SLS) solvers. Their experimental results show that SLS solvers benefit the present of resolution-based preprocessing and multiple preprocessing techniques. And the second one is that of Lynce and Marques-Silva [LMS01]. They only evaluated empirically the impact of some preprocessors developed before 2001 including 3-Resolution, without considering multiple preprocessing, on the performance of systematic SAT solvers. In recent years, many other preprocessors, which are sophisticated, have been applied to modern propositional reasoners. Among them are 2-SIMPLIFY [Bra01], the preprocessor in Lsat [OGMS02] for recovering and exploiting Boolean gates, HyPre [Bac02, BW04], Shatter [ASM03] for dealing with symmetry structure, NiVER [SP05] and SatELite [EB05]. We consider some of these preprocessors plus 3-Resolution in our study.

3 SAT Preprocessors

We describe briefly the six SAT preprocessors used in the experiments. The first five are all based on resolution and its variants such as hyper-resolution. Resolution [Qui55, DP60, Rob65] itself is widely used as a rule of inference in first order automated deduction, where the clauses tend to be few in number and contain few literals, and where the reasoning is primarily driven by unification. As a procedure for propositional reasoning, however, resolution is rarely used on its own because in practice it has not been found to lead to efficient algorithms. The sixth preprocessor is a special-purpose tool for symmetry detection, which is important for one problem class in the experiments.

3.1 3-Resolution

k-Resolution is just saturation under resolution with the restriction that the parent clauses are of length at most k. The special cases of 2-Resolution and 3-Resolution are of most interest. 3-Resolution has been used in a number of SAT solvers, notably Satz [LA97] and the SLS solver R+AdaptNovelty$^+$ [APSS05] which won the satisfiable random problem category in the SAT2005 competition. Since it is the preprocessor already used by Satz, we expect it to work well with Dew_Satz.

3.2 2-SIMPLIFY

2-SIMPLIFY [Bra01] constructs an implication graph from all binary clauses in the problem. Where there is an implication chain from a literal X to $\overline{X}$, $\overline{X}$ can be deduced

as a unit which can be propagated. The method also collapses strongly connected components, propagates *shared implications*, or literals implied in the graph by every literal in a clause, and removes some redundant binary clauses. Experimental results [Bra01, Bra04] show that systematic search benefits markedly from 2-SIMPLIFY on a wide range of problems.

3.3 HyPre

HyPre [BW04] also reasons with binary clauses, but incorporates full hyper-resolution, making it more powerful than 2-SIMPLIFY. In addition, unit reduction and equality reduction are incrementally applied to infer more binary clauses. It can be costly in terms of time, but since it is based explicitly on hyper-resolution it avoids the space explosion of computing a full transitive closure. HyPre has been used in the SAT solver, 2CLS+EQ [Bac02], and we consider it a very promising addition to many other solvers. It is generally useful for exploiting implicational structure in large problems.

3.4 NiVER

Variable Elimination Resolution (VER) is an ancient inference method consisting of performing all resolutions on a chosen variable and then deleting all clauses in which that variable occurs, leaving just the resolvents. It is easy to see that this is a complete decision procedure for SAT problems, and almost as easy to see that it is not practicable because of exponential space complexity. Recently, Subbarayan and Pradhan [SP05] proposed NiVER (Non increasing VER) which restricts the variable elimination to the case in which there is no increase in the number of literals after elimination. This shows promise as a SAT preprocessor, improving the performance of a number of solvers [SP05].

3.5 SatELite

Eén and Biere [EB05] proposed the SatELite preprocessor, which extends NiVER with a rule of Variable Elimination by Substitution. Several additions including subsumption detection and improved data structures further improved performance in both space and time. SatELite was combined with MiniSat to form SatELiteGTI, the system which dominated the SAT2005 competition on the crafted and industrial problem categories. Since we use MiniSat for our experiments, it is obvious that SatELite should be one of the preprocessors we consider.

3.6 Shatter

It is clear that eliminating symmetries is essential to solving realistic instances of many problems. None of the resolution-based preprocessors does this, so for problems that involve a high degree of symmetry we added Shatter [AMS03] which detects symmetries and adds symmetry-breaking clauses. These always increase the size of the clause set and for satisfiable problems they remove some of the solutions, but they typically make the problem easier by pruning away isomorphic copies of parts of the search space.

4 SAT Solvers

As noted in Section 1, we concentrate on just two solvers: MiniSat, which relies on clause learning, and Dew_Satz, which uses lookahead.

4.1 MiniSat

Sörensson and Eén [ES03, SE05] released the MiniSat solver in 2005. Its design is based on Chaff, particularly in that it learns nogoods or "conflict clauses" and accesses them during the search by means of two watched literals in each clause. MiniSat is quite small (a few hundred lines of code) and easy to use either alone or as a module of a larger system. Its speed in comparison with similar solvers such as zChaff comes from a series of innovations of which the most important are an activity-decay schedule which proceeds by frequent small reductions rather than occasional large ones, and an inference rule for reducing the size of conflict clauses by introducing a restricted subsumption test. The cited paper contains a brief but informative description of these ideas.

4.2 Dew_Satz

The solver Dew_Satz [AS05] is a recent version of the Satz solver [LA97]. Like its parent Satz, it gains efficiency by a restricted one-step lookahead scheme which rates some of the neighbouring variables every time a choice must be made for branching purposes. Its lookahead is more sophisticated than the original one of Satz, adding a DEW (dynamic equality weighting) heuristic to deal with equalities. This enables the variable selection process to avoid duplicating the work of weighting variables detected to be equivalent to those already examined. Thus, while the solver has no special inference mechanism for propositional equalities, it does deal tolerably well with problems containing them.

5 Experimental Results

We present results on four benchmark problem sets chosen to present challenges for one or other or both of the SAT solvers. The experiments were conducted on a cluster of 16 AMD Athlon 64 processors running at 2 GHz with 2 GB of RAM. Ptime in the tables represents preprocessing time, while Stime represents solvers runtime without including Ptime. The timebound of Stime is 15,000 seconds per problem instance. It is worth noting that in our study the results of SatELiteGTI, the solver which dominated the SAT2005 competition on the crafted and industrial problem categories, are represented by the results of SatELite+MiniSat.

5.1 The 32-bit Parity Problem

The 32-bit parity problem was listed by Selman *et al.* [SKM97] as one of ten challenges for research on satisfiability testing. The ten instances of the problem are satisfiable. The first response to this challenge was by Warners and van Maaren [WvM98] who solved the `par32-*-c` problem (5 instances) using a special-purpose preprocessor to deal with equivalency conditions. Two years later, Li [Li00] solved the ten `par32*` instances by enhancing Satz's search process with equivalency reasoning. Ostrowski *et*

Instance	Prep.	#Vars/#Cls/#Lits	Ptime	Dew_Satz		MiniSat	
				Stime	#BackT	Stime	#Conflict
par32-1	Orig	3176/10227/27501	n/a	>15,000	n/a	>15,000	n/a
	3Res	2418/7463/19750	0.08	12,873	17,335,530	>15,000	n/a
	Hyp+3Res	1313/6193/17203	0.50	9,513	17,391,333	>15,000	n/a
	Niv+3Res	1315/5948/16707	0.46	6,858	13,476,105	>15,000	n/a
	3Res+Hyp	1313/5495/15810	0.11	9,655	17,335,492	>15,000	n/a
	Sat+3Res	849/5245/18660	0.37	14,729	34,569,968	>15,000	n/a
par32-2	Orig	3176/10253/27405	n/a	>15,000	n/a	5,364	9,125,821
	3Res	2392/7387/19550	0.08	5,171	9,341,185	6,205	10,492,612
	Hyp+3Res	1301/5975/16719	0.36	3,831	8,186,883	>15,000	n/a
	Niv+3Res	1303/5730/16223	0.29	1,518	3,889,345	>15,000	n/a
par32-3	Orig	3176/10297/27581	n/a	>15,000	n/a	>15,000	n/a
	3Res	2395/7437/19738	0.07	6,124	9,711,576	>15,000	n/a
	Hyp+3Res	1323/5961/16779	0.23	3,673	9,708,520	>15,000	n/a
	Niv+3Res	1325/5716/16283	0.22	4,470	9,710,552	>15,000	n/a
	Sat+3Res	848/5284/18878	0.37	3,647	2,206,369	>15,000	n/a
par32-4	Orig	3176/10313/27645	n/a	>15,000	n/a	>15,000	n/a
	3Res	2385/7433/19762	0.08	10,425	10,036,154	>15,000	n/a
	Sat	849/5160/18581	0.21	12,820	18,230,746	>15,000	n/a
	Hyp+3Res	1331/6055/16999	0.36	9,001	17,712,997	>15,000	n/a
	3Res+Hyp	1331/5567/16026	0.11	5,741	10,036,146	>15,000	n/a
	Niv+3Res	1333/5810/16503	0.34	6,099	10,036,154	>15,000	n/a
	3Res+Niv	1290/5297/15481	0.10	14,003	25,092,756	>15,000	n/a
	3Res+Sat	850/5286/18958	0.35	3,552	7,744,986	>15,000	n/a
	Sat+3Res	849/5333/19052	0.38	3,563	7,744,986	>15,000	n/a
	Sat+2Sim	848/5154/18565	0.26	12,862	18,230,746	>15,000	n/a
par32-5	Orig	3176/10325/27693	n/a	>15,000	n/a	>15,000	n/a
	Niv	1978/7864/22535	0.03	10,651	27,165,469	>15,000	n/a
par32-1-c	Orig	1315/5254/15390	n/a	>15,000	n/a	>15,000	n/a
	3Res	1315/5957/16738	0.35	11,068	25,920,943	>15,000	n/a
	Hyp+3Res	1313/6193/17203	0.48	7,419	8,931,149	>15,000	n/a
par32-2-c	Orig	1303/5206/15246	n/a	>15,000	n/a	>15,000	n/a
	3Res	1303/5739/16254	0.23	428	345,680	>15,000	n/a
	Hyp+3Res	1301/5975/16719	0.32	7,402	8,166,758	>15,000	n/a
par32-3-c	Orig	1325/5294/15510	n/a	>15,000	n/a	>15,000	n/a
	3Res	1325/5725/16314	0.15	4,482	9,462,205	>15,000	n/a
	Hyp	1323/5589/16094	0.04	11,745	19,947,965	>15,000	n/a
	Hyp+3Res	1323/5961/16779	0.22	4,375	9,462,245	>15,000	n/a
	Niv+3Res	1321/5708/16266	0.24	7,361	16,265,438	>15,000	n/a
	Sat+3Res	802/5335/19335	0.33	5,407	7,280,963	>15,000	n/a
par32-4-c	Orig	1333/5326/15606	n/a	>15,000	n/a	>15,000	n/a
	3Res	1333/5819/16534	0.24	7,097	8,440,212	>15,000	n/a
	Hyp+3Res	1331/6055/16999	0.32	5,175	9,669,012	>15,000	n/a
	Niv+3Res	1329/5802/16486	0.55	10,495	21,738,376	>15,000	n/a
	Sat+3Res	806/5357/19443	0.32	10,110	8,013,977	>15,000	n/a
par32-5-c	Orig	1339/5350/15678	n/a	10,949	22,878,571	>15,000	n/a
	Niv+3Res	1335/5728/16362	0.28	>15,000	n/a	7,363	16,189,524

Table 1: Dew_Satz and MiniSat performance, before and after preprocessing, on par32 problem.

al. [OGMS02], solved the problems with Lsat, which performs a preprocessing step to recover and exploit the logical gates of a given CNF formula and then applies DPLL with a Jeroslow-Wang branching rule. The challenge has now been met convincingly by Heule *et al.* [HvM04] with their March_eq solver, which combines equivalency reasoning in a preprocessor with a lookahead-based DPLL and which solves all of the `par32*` instances in seconds. Dew_Satz is one of the few solvers to have solved any instances of the 32-bit parity problem without special-purpose equivalency reasoning [AS05].

Table 1 shows the results of running the lookahead-based solver Dew_Satz and the CDCL-based solver MiniSat on the ten `par32` instances, with and without preprocessing. As preprocessors we used 3-Resolution, HyPre, NiVER and SatELite alone and followed by 3-Resolution for the last three. We eliminated 2-SIMPLIFY from this test as it aborted the resolution process of the first five `par32*` instances presented in the Table 1. We experimented also with all combination of two preprocessors for the problems `par32-1` and `par32-4`. Where lines are omitted from the table (e.g. there is no line for HyPre on `par32-1` and for SatELite+3-Resolution on `par32-2`), this is because no single solver produced a solution for those simplified instances.

It is evident from the table that these problems are seriously hard for both solvers. Even with preprocessing, MiniSat times out on all of them except for `par32-2` and `par32-5-c`. Curiously, on `par32-2` instance, preprocessing with 3-Resolution makes its performance degrade a little. This is not a uniform effect: Table 4 below shows examples in which MiniSat benefits markedly from 3-Resolution. Without preprocessing, Dew_Satz times out on nine of ten `par32` instances, but in every case except `par32-5` and `par32-5-c` 3-Resolution suffices to help it find a solution, and running multiple preprocessors improves its performance.

In general, Table 1 shows that multiple preprocessing contributes significantly to enhance the performance of Dew_Satz and the preprocessor 3-Resolution dominates the contribution through either single or multiple preprocessing.

5.2 A Planning Benchmark Problem

The `ferry` planning benchmark problems, taken from SAT2005 competition, are all easy for MiniSat, which solves all of them in about one second without needing preprocessors. Dew_Satz, however, is challenged by them. The problems are satisfiable. We show the Dew_Satz and MiniSat results on the problems in Table 2. Clearly the original problems contain some structure that CDCL is able to exploit but which is uncovered by one-step lookahead. It is therefore interesting to see which kinds of reasoning carried out in a preprocessing phase are able to make that same structure available to Dew_Satz. Most strikingly, reasoning with binary clauses in the manner of the 2-SIMPLIFY preprocessor reduces runtimes by upwards of four orders of magnitude in some cases. HyPre, NiVER and SatELite, especially HyPre, are also effective on these planning problems. In most cases the number of backtracks reduces from million to less than 100 or even zero for `ferry8_v01a`, `ferry9_v01a`, and `ferry10_ks99a` instances which means that the input formula is solved at the root node of the search tree.

Instance	Prep.	#Vars/#Cls/#Lits	Ptime	Dew_Satz		MiniSat	
				Stime	#BackT	Stime	#Conflict
ferry7_ks99i	Orig	1946/22336/45706	n/a	2,828	10,764,261	0.13	4,266
	3Res	1930/22289/45621	0.09	>15,000	n/a	0.11	3,707
	Hyp	1881/32855/66732	0.21	1,672	1,204,321	0.03	417
	Niv	1543/21904/45243	0.01	>15,000	n/a	0.10	3,469
	Sat	1286/21601/50644	0.33	>15,000	n/a	0.07	2,763
	Sat+2Sim	1279/56597/120318	0.49	0.41	28	0.05	1,096
ferry7_v01i	Orig	1329/21688/50617	n/a	>15,000	n/a	0.05	1,858
	3Res	1329/21681/50505	0.14	>15,000	n/a	0.05	1,858
	Sat	1286/21609/50803	0.17	>15,000	n/a	0.18	6,309
	Sat+2Sim+3Res	1286/64472/136299	0.95	4.28	824	0.05	1,018
	Sat+2Sim+Hyp+3Res	1272/62208/131357	1.26	3.30	580	0.10	2,398
ferry8_ks99a	Orig	1259/15259/31167	n/a	574	1,869,995	0.01	0
	3Res	1241/15206/31071	0.08	654	2,074,794	0.01	0
	Sat	813/14720/34687	0.24	810	1,040,528	0.01	381
	Sat+2Sim	813/35008/75263	0.35	0.11	4	0.02	295
ferry8_ks99i	Orig	2547/32525/66425	n/a	>15,000	n/a	0.22	6,615
	3Res	2529/32472/66329	0.12	>15,000	n/a	0.14	3,495
	Hyp	2473/48120/97601	0.32	>15,000	n/a	0.07	1,030
	Sat	1696/31589/74007	0.49	>15,000	n/a	0.41	10,551
	Sat+2Sim	1683/83930/178217	0.76	9.38	3,255	0.20	5,105
ferry8_v01a	Orig	854/14819/34624	n/a	13,162	39,153,348	0.01	277
	3Res	854/14811/34480	0.11	>15,000	n/a	0.01	277
	Hyp	846/38141/81268	0.18	6.11	570	0.02	226
	Hyp+Sat	813/38044/81364	0.36	29.66	2,749	0.02	277
	Hyp+Sat+3Res	813/38028/81196	0.70	0.71	15	0.02	277
	Hyp+Sat+2Sim	813/36583/78442	0.50	0.17	0	0.02	233
ferry8_v01i	Orig	1745/31688/73934	n/a	>15,000	n/a	0.55	12,935
	3Res	1745/31680/73790	0.20	>15,000	n/a	0.55	12,935
	Sat	1696/31598/74202	0.25	>15,000	n/a	0.25	7,869
	Sat+2Sim+3Res	1696/96092/202904	1.50	268	68,681	4.06	28,690
ferry9_ks99a	Orig	1598/21427/43693	n/a	>15,000	n/a	0.01	0
	3Res	1578/21368/43586	0.10	>15,000	n/a	0.01	0
	Hyp	1542/29836/60522	0.21	>15,000	n/a	0.02	278
	Niv	1244/21046/43264	0.01	>15,000	n/a	0.01	29
	Sat	1042/20765/48878	0.33	>15,000	n/a	0.02	350
	2Sim	1569/20563/41976	0.02	>15,000	n/a	0.04	1,359
	Hyp+Sat	1056/26902/72553	0.88	33.73	22,929	0.03	609
	Sat+2Sim	1042/50487/108322	0.50	0.18	5	0.03	261
ferry9_v01a	Orig	1088/20878/48771	n/a	>15,000	n/a	0.01	181
	3Res	1088/20869/48591	0.16	>15,000	n/a	0.01	181
	Hyp	1079/55371/117757	0.28	0.42	0	0.03	187
	Hyp+Sat	1042/55256/117861	0.49	70.83	5,080	0.03	181
	Hyp+Sat+2Sim	1042/53394/114135	0.72	0.39	2	0.03	234
ferry10_ks99a	Orig	1977/29041/59135	n/a	>15,000	n/a	0.03	710
	3Res	1955/28976/59017	0.13	>15,000	n/a	0.03	827
	Hyp	1915/40743/82551	0.29	>15,000	n/a	0.04	563
	Niv	1544/28578/58619	0.02	>15,000	n/a	0.01	0
	Sat	1299/28246/66432	0.44	>15,000	n/a	0.03	909
	2Sim	1945/27992/57049	0.05	>15,000	n/a	0.05	1,565
	Sat+2Sim	1299/69894/149728	0.69	0.28	1	0.04	419
	3Res+2Sim+Niv	1793/21099/43369	0.43	0.08	0	0.06	1,278
	Niv+Hyp+2Sim+3Res	1532/24524/50463	0.54	5.19	3,949	0.02	454
ferry10_v01a	Orig	1350/28371/66258	n/a	>15,000	n/a	0.02	191
	3Res	1350/28361/66038	0.23	>15,000	n/a	0.02	191
	Hyp	1340/77030/163576	0.40	4.90	550	0.04	401
	Hyp+Sat+3Res	1299/76874/163442	1.56	1,643	118,635	0.04	343
	Hyp+Sat+2Sim	1299/74615/159134	1.00	1.78	61	0.04	459

Table 2: Dew_Satz and MiniSat performance, before and after preprocessing, on ferry planning problem.

Instance	Prep.	#Vars/#Cls/#Lits	Ptime	Dew_Satz		MiniSat	
				Stime	#BackT	Stime	#Conflict
bmc-ibm-3	Orig	14930/72106/189182	n/a	>15,000	n/a	0.39	1,738
	Hyp	5429/32038/89471	3.72	113	2,327	0.06	379
	Niv	10591/62966/176261	0.47	>15,000	n/a	0.36	2,088
	3Res	11940/56736/148383	0.41	>15,000	n/a	0.37	2,374
	Sat	6486/44239/137653	1.32	>15,000	n/a	0.21	1,744
	Hyp+3Res	5429/30517/79041	3.98	19.67	68	0.04	206
bmc-galileo-8	Orig	58073/294821/767187	n/a	>15,000	n/a	0.37	462
	Hyp	9613/85311/202625	416	67.92	0	0.06	2
	Niv	30788/240141/685499	1.06	>15,000	n/a	0.42	1,148
	3Res	43962/182261/456906	5.91	>15,000	n/a	0.46	1,182
	Sat	20593/135076/414093	6.46	>15,000	n/a	0.19	762
	Hyp+3Res	9613/82572/188966	417	120	0	0.05	2
	Sat+3Res	20561/134793/413048	10.12	19.00	1	0.18	799
bmc-galileo-9	Orig	63623/326999/852078	n/a	>15,000	n/a	0.54	1,186
	Hyp	8802/70198/170970	407	90.50	0	0.06	2
	Niv	33872/267378/763037	1.17	>15,000	n/a	0.42	1,148
	3Res	49400/208310/523628	6.56	>15,000	n/a	0.46	1,060
	Sat	23381/155837/477951	7.47	>15,000	n/a	0.24	1,009
	Hyp+3Res	8802/67042/155884	408	57.57	0	0.04	2
bmc-ibm-10	Orig	59056/323700/854093	n/a	>15,000	n/a	1.77	4,277
	Hyp	2259/10831/29155	47.28	0.30	0	0.01	0
	Niv	40530/285198/797443	1.19	>15,000	n/a	0.90	3,532
	3Res	32377/154730/400447	4.70	>15,000	n/a	1.32	3,276
	Sat	14956/116772/404199	5.53	>15,000	n/a	0.49	2,502
bmc-ibm-11	Orig	32109/150027/394770	n/a	>15,000	n/a	2.01	6,422
	Hyp	7342/40802/106327	13.84	4.73	1	0.05	160
	Niv	22927/130058/365568	0.96	>15,000	n/a	1.33	5,607
	3Res	22709/98066/252495	1.46	>15,000	n/a	2.44	7,875
	Sat	10071/62668/200137	2.58	>15,000	n/a	0.77	5,481
bmc-ibm-12	Orig	39598/194778/515536	n/a	>15,000	n/a	8.41	11,887
	Hyp	12205/87082/228241	91.61	>15,000	n/a	0.74	1,513
	Niv	27813/168440/476976	0.69	>15,000	n/a	4.46	8,702
	3Res	32606/160555/419341	2.77	>15,000	n/a	6.77	10,243
	Sat	15176/109121/364968	4.50	>15,000	n/a	2.37	6,219
	Niv+Hyp+ 3Res	12001/100114/253071	85.81	106	6	0.76	1,937
bmc-ibm-13	Orig	13215/65728/174164	n/a	>15,000	n/a	1.84	8,088
	Hyp	5010/27248/78059	3.16	>15,000	n/a	0.13	1,018
	Niv	9226/57332/161962	0.35	>15,000	n/a	1.72	9,181
	3Res	10426/49594/129998	0.49	>15,000	n/a	13.17	30,687
	Sat	4549/34273/110676	1.27	>15,000	n/a	1.25	9,324
	3Res+Niv+ Hyp+3Res	3529/22589/62633	2.90	1,575	4,662,067	0.03	150
bmc-alpha-25449	Orig	663443/3065529/7845396	n/a	>15,000	n/a	6.64	502
	Sat	12408/76025/247622	129	6.94	7	0.06	1
	Sat+Hyp	9091/61789/203593	566	7.82	2	0.10	109
	Sat+Niv	12356/75709/246367	130	4.48	2	0.06	1
	Sat+3Res	12404/77805/249192	130	8.84	1	0.06	1
	Sat+2Sim	10457/71128/229499	131	6.37	10	0.10	133
bmc-alpha-4408	Orig	1080015/3054591/7395935	n/a	>15,000	n/a	5,409	587,755
	Sat	23657/112343/364874	47.22	>15,000	n/a	1,266	820,043
	Sat+Hyp	13235/88976/263053	56.13	>15,000	n/a	8,753	4,916,981
	Sat+Niv	22983/108603/351369	49.34	>15,000	n/a	2,137	1,294,590
	Sat+3Res	23657/117795/380389	48.18	>15,000	n/a	946	618,853
	Sat+2Sim	17470/129245/375444	51.55	>15,000	n/a	804	561,529
	Sat+2Sim+ 3Res	16837/98726/305057	52.89	>15,000	n/a	571	510,705

Table 3: Dew_Satz and MiniSat performance, before and after preprocessing, on hard BMC instances.

Instance	Prep.	#Vars/#Cls/#Lits	Ptime	Dew_Satz		MiniSat	
				Stime	#BackT	Stime	#Conflict
01-k10	Orig	9275/38802/98468	n/a	18.96	1,472	0.08	313
	Hyp	n/a	1.27	n/a	n/a	n/a	n/a
	Niv	6662/33394/90715	0.16	4.12	366	0.07	327
	3Res	6498/27318/70158	0.24	26.38	2,860	0.07	282
	Sat	3418/19648/62925	0.84	3.46	140	0.03	262
	2Sim	4379/59765/133585	3.41	0.24	1	0.05	135
01-k15	Orig	11524/48585/123966	n/a	>15,000	n/a	0.49	3,743
	3Res+Sat+ Niv+Hyp	3382/25936/79364	4.65	1,420	130,013	0.06	682
	3Res+Hyp+ Niv+3Res	4203/23731/60639	4.33	190	13,449	0.06	430
	3Res+Hyp+ 3Res	4732/24972/63133	4.17	262	19,701	0.04	243
	Hyp	4889/27056/71819	3.94	2,937	178,245	0.06	603
	Niv	8068/41368/113317	0.19	>15,000	n/a	0.49	3,548
	3Res	9403/40059/103137	0.27	>15,000	n/a	0.66	3,783
	Sat	5198/30697/97961	1.13	>15,000	n/a	0.32	3,655
01-k20	Orig	15069/63760/163081	n/a	>15,000	n/a	4.95	16,658
	3Res+Hyp+ Niv+3Res	6382/34846/89807	6.94	513	29,629	0.20	1,261
	Hyp	7323/39150/104635	6.40	>15,000	n/a	0.95	5,182
	Niv	10533/54293/149192	0.25	>15,000	n/a	0.28	2,069
	3Res	12948/55490/142966	0.37	>15,000	n/a	1.58	9,341
	Sat	7179/42837/136537	1.52	>15,000	n/a	1.11	8,705
	2Sim	9370/93921/217635	1.91	>15,000	n/a	0.35	1,977
26-k70	Orig	346561/1752741/4579945	n/a	>15,000	n/a	8,382	2,654,614
	3Res	346561/1756001/4588705	150	21.32	1	1.02	10
	Hyp	243461/1569549/4182061	338	>15,000	n/a	1.22	642
	Niv	155221/1354556/4075072	479	>15,000	n/a	1.07	492
	Sat	132670/1300914/4980854	109	>15,000	n/a	2,325	1,503,271
26-k75	Orig	371091/1877066/4904440	n/a	>15,000	n/a	8,540	2,880,376
	3Res	371091/1880536/4913780	161	22.81	1	1.06	11
	Hyp	260621/1680704/4477966	364	>15,000	n/a	1.20	654
	Niv	166195/1450679/4364543	4.95	>15,000	n/a	1.41	474
	Sat	141870/1392526/5327557	117	>15,000	n/a	3,896	2,067,948
26-k85	Orig	420151/2125716/5553430	n/a	>15,000	n/a	>15,000	n/a
	3Res	420151/2129606/5563930	183	25.43	1	1.21	10
	Hyp	294941/1903014/5069776	417	>15,000	n/a	1.55	747
	Niv	187631/1641901/4941437	5.69	>15,000	n/a	1.37	535
	Sat	160270/1576770/6039510	132	>15,000	n/a	4,472	2,308,225
26-k90	Orig	444681/2250041/5877925	n/a	>15,000	n/a	>15,000	n/a
	Niv+3Res	198605/2208074/6624608	121	13.21	1	1.77	5
	Hyp+3Res	312101/1979389/5187311	600	46.53	1	1.39	5
	3Res	444681/2254141/5889005	195	26.99	1	1.26	10
	Hyp	312101/2014169/5365681	446	>15,000	n/a	1.55	583
	Niv	198605/1738024/5230908	5.91	>15,000	n/a	1.38	429
	Sat	169470/1669436/6402318	140	>15,000	n/a	8,240	3,311,629

Table 4: Dew_Satz and MiniSat performance, before and after preprocessing, on SAT2005 IBM-FV-* instances.

5.3 Bounded Model Checking Problems

Another domain providing benchmark problem sets which appear to be easy for MiniSat but sometimes hard for Dew_Satz is bounded model checking. In Table 3 we report results on five of eleven `BMC-IBM` problems, two `BMC-galileo` problems and two of four `BMC-alpha` problems. All other benchmark problems in the `BMC-IBM` class are easy for both solvers and so are omitted from the table. The other two `BMC-alpha` instances are harder than the two reported even for MiniSat before and after preprocessing. The problems presented in Table 3 are satisfiable.

Each of these bounded model checking problems is brought within the range of Dew_Satz by some form of preprocessing. In general, HyPre and 3-Resolution are the best for this purpose, especially when used together, though on problem `BMC-IBM-13` they are ineffective without the additional use of NiVER. The column showing the number of times Dew_Satz backtracks is worthy of note. In many cases, preprocessing reduces the problem to one that can be solved without backtracking. Solving "without backtracking" has to be interpreted with care here, of course, since a nontrivial amount of lookahead may be required in a "backtrack-free" search. The results for `BMC-galileo-9` furnish a good example of this: HyPre takes 407 seconds to refine the problem, following which Dew_Satz spends 90 seconds on lookahead reasoning while constructing the first (heuristic) branch of its search tree, but then that branch leads directly to a solution. Adding 3-Resolution to the preprocessing step does not change the number of variables, and only slightly reduces the number of clauses, but it roughly halves the time subsequently spent on lookahead.

The instance `BMC-alpha-4408` is hard for Dew_Satz even after preprocessing. While MiniSat with multiple preprocessing solves the problem instance with an order of magnitude faster. We can also observe that HyPre brings more benefit than SatELite,

Table 4 shows results for both solvers on a related problem set consisting of formal verification problems taken from the SAT2005 competition. The `IBM-FV-01` problems are satisfiable except for the problem `IBM-FV-01-k10`; the `IBM-FV-26` problems are unsatisfiable. Most of these satisfiable problems are easy for MiniSat, but the unsatisfiable cases show that the SatELite preprocessor (with which MiniSat was paired in the competition) is by far the least effective of the four we consider for MiniSat on these problems. The preprocessor HyPre proved the unsatisfiability of `IBM-FV-01-k10` in 1.27 seconds. 2-SIMPLIFY was not used to simplify the `IBM-FV-26` problems, because it is limited for input formula with maximum 100,000 variables. Again there are cases in which Dew_Satz is improved from a 15,000 second timeout to a one-branch proof of unsatisfiability. Note that the numbers of clauses in these cases are actually increased by the preprocessor 3-Resolution, confirming that the point of such reasoning is to expose structure rather than to reduce problem size.

5.4 A Highly Symmetrical Problem

FPGA routing problem is a higly symmetrical problem that model the routing of wires in the channels of field-programmable integrated circuits [AMS03]. The problem instances used in the experiment, which were artificially designed by Fadi Aloul, are taken from SAT2002 competition.

Without preprocessing to break symmetries, many of the FPGA routing problems are hard—harder for CDCL solvers than for lookahead-based ones. Not only do they have many symmetries, but the clause graphs are also disconnected. Lookahead techniques with neighbourhood variables ordering heuristic seem able to choose inferences within one graph component before moving to another, whereas MiniSat jumps frequently between components. Table 5 shows performances of both solvers on FPGA routing problem set. Of 21 selected satisfiable (`bart`) problems, MiniSat solves 8 in some 2 hours. It manages better with the unsatisfiable (`homer`) instances, solving 14 of 15 in a total time of around 6 hours. Dew_Satz solves all of the `bart` problems in 17.5 seconds and the `homer` ones in 45 minutes.

The detailed results for two of the satisfiable problems and two unsatisfiable ones (Table 6) are interesting. The resolution-based preprocessors do not give any modification to the size of the input formula except when using SatELite. The Shatter preprocessor, which removes certain symmetries, is tried on its own and in combination with the five resolution-based preprocessors. It should be noted that the addition of symmetry-breaking clauses increases the sizes of the problems, but of course it greatly reduces the search spaces in most cases.

The performance of Dew_Satz after preprocessing is often worse in terms of time than it was before, though there is always a decreases in the size of its search tree. This is because of the increase in the problem size which increases the amount of lookahead process. MiniSat, by contrast, sometimes speeds up by several orders of magnitude after preprocessing.

Instance	Dew_Satz			MiniSat		
	#Solved	Stime	#BackT	#Solved	Stime	#Conflict
bart (21 SAT)	21	17.52	1,536,966	8	7,203	119,782,466
homer (15 UNSAT)	15	2,662	109,771,200	14	22,183	143,719,166

Table 5: Dew_Satz and MiniSat performance, without preprocessing, on FPGA routing problems.

5.5 Order of Preprocessors

Table 7 illustrates the difficulty of selecting the order in which to apply multiple preprocessors. It shows results on just two sample problems. The first is the bounded model checking problem `BMC-IBM-12`, which Dew_Satz attempted with the three preprocessors HyPre, NiVER and 3-Resolution in different orders. Only one order, NiVER followed by HyPre followed by 3-Resolution, renders the problem feasible for Dew_Satz. With the preprocessors in that order, it is solved in less than 2 minutes; with any other order it cannot be solved in more than four hours. The second problem, `ferry10_ks99a`, shows the range of different outcomes produced by varying the order of four preprocessors. If we get it right, we get a solution in 5 seconds, but we know of no simple rule for getting it right in such a case. Neither running NiVER first nor running 3-Resolution last is sufficient. Even with NiVER, HyPre and 3-Resolution in the right order, putting 2-SIMPLIFY first rather than third changes the runtime from 5 seconds to several hours. The third experiment illustrates the efffect of alternating two preprocessors. Simplifying

Instance	Prep.	#Vars/#Cls/#Lits	Ptime	Dew_Satz		MiniSat	
				Stime	#BackT	Stime	#Conflict
bart28	Orig	428/2907/7929	n/a	0.00	0	>15,000	n/a
	Sat	413/2892/11469	0.06	0.02	0	>15,000	n/a
	Sha	1825/8407/27003	0.37	0.06	9	198	775,639
	Sha+3Res	1764/7702/24400	0.46	0.04	1	2,458	7,676,459
	Sha+Hyp	1764/8349/26138	0.41	0.05	20	>15,000	n/a
	Sha+Niv	1781/8358/26759	0.38	0.05	6	5.46	53,683
	Sha+Sat	1728/8254/30422	0.53	0.10	0	115	684,272
	Sha+2Sim	1750/7892/24682	0.39	0.05	17	19.12	150,838
bart30	Orig	485/3617/9954	n/a	0.31	20,160	>15,000	n/a
	Sat	468/3600/14544	0.08	0.03	0	>15,000	n/a
	Sha	2017/9649/30874	0.49	0.11	96	>15,000	n/a
	Sha+3Res	1945/8686/27492	0.60	0.12	224	4,149	7,594,231
	Sha+Hyp	1945/9348/29218	0.54	11,729	28,270,212	>15,000	n/a
	Sha+Niv	1969/9599/30625	0.50	0.06	1	>15,000	n/a
	Sha+Sat	1776/8830/33533	0.77	0.12	1	>15,000	n/a
	Sha+2Sim	1919/8758/27287	0.51	0.05	9	>15,000	n/a
homer19	Orig	330/2340/4950	n/a	473	19,958,400	10,233	51,960,410
	Sat	300/2310/8400	0.04	>15,000	n/a	5,621	54,469,568
	Sha	1460/6764/20242	0.16	2,345	4,828,639	2.26	33,492
	Sha+3Res	1388/5748/16914	0.23	3,231	7,189,966	1.14	21,669
	Sha+Hyp	1387/6547/18865	0.20	4,179	9,611,768	1.90	30,418
	Sha+Niv	1412/6715/19993	0.17	2,570	5,202,084	3.15	45,484
	Sha+Sat	1201/5846/19288	0.34	4,071	6,236,966	1.48	26,517
	Sha+2Sim	1348/5639/16110	0.17	307	678,425	0.70	14,682
homer20	Orig	440/4220/8800	n/a	941	19,958,400	>15,000	n/a
	Sat	400/4180/15200	0.08	1,443	6,982,425	11,448	57,302,582
	Sha	1999/10340/29988	0.28	369	350,610	1.83	22,950
	Sha+3Res	1907/8793/25027	0.37	362	405,059	1.41	18,273
	Sha+Hyp	1905/10527/29129	0.34	1,306	1,451,567	1.10	13,927
	Sha+Niv	1941/10276/29671	0.29	379	349,842	0.91	13,543
	Sha+Sat	1723/9420/30986	0.54	822	300,605	1.00	13,831
	Sha+2Sim	1879/9419/26188	0.31	114	120,297	0.40	6,612

Table 6: Dew_Satz and MiniSat performance, before and after preprocessing, on selected FPGA routing instances.

Instance	Prep.	#Vars/#Cls/#Lits	Ptime	Stime	#BackT
bmc-ibm-12	Hyp+3Res+Niv	10805/83643/204679	96.11	>15,000	n/a
	Niv+Hyp+3Res	12001/100114/253071	85.81	106	6
	3Res+Hyp+Niv	10038/82632/221890	89.56	>15,000	n/a
	3Res+Niv+Hyp	11107/99673/269405	58.38	>15,000	n/a
ferry10_ks99a	2Sim+Niv+Hyp+3Res	1518/32206/65806	0.43	>15,000	n/a
	Niv+3Res+2Sim+Hyp	1532/25229/51873	0.49	11,345	17,778,483
	3Res+2Sim+Niv+Hyp	1793/20597/42365	0.56	907	1,172,964
	Niv+Hyp+2Sim+3Res	1532/24524/50463	0.54	5.19	3,949
ferry10_ks99a	2Sim+Niv	1518/27554/56565	0.08	>15,000	n/a
	2Sim+Niv+2Sim	1518/18988/39433	0.27	3,197	6,066,241
	2Sim+Niv+2Sim+Niv	1486/18956/39429	0.29	129	290,871
	2Sim+Niv+2Sim+Niv+2Sim	1486/23258/48033	0.48	7,355	8,216,100

Table 7: Dew_Satz's performance on instances with preprocessor ordering.

with 2-SIMPLIFY followed by NiVER is insufficient to allow solution before the timeout. Simplifying again with 2-SIMPLIFY brings the runtime down to under an hour; adding NiVER again brings it down again to a couple of minutes; repeating 2-SIMPLIFY, far from improving matters, causes the time to blow out to two hours.

6 Conclusions

We performed an empirical study of the effects of several recently proposed SAT preprocessors on both CDCL and lookahead-based SAT solvers. We describe several outcomes from this study as follow.

1. High-performance SAT solvers, whether they depend on clause learning or on lookahead, benefit greatly from preprocessing. Improvements of four orders of magnitude in runtimes are not uncommon.

2. It is unlikely to equip a SAT solver with just one preprocessor of the kind considered in this paper. Very different preprocessing techniques are appropriate to different problem classes.

3. There are frequently benefits to be gained from running two or more preprocessors in series on the same problem instance.

4. Both clause learning and lookahead need to be enhanced with techniques specific to reasoning with binary clauses, in order to exploit dependency chains, and with techniques for equality reasoning.

5. Lookahead-based solvers also benefit greatly from resolution between longer clauses, as in the 3-Resolution preprocessor. This seems to capture ahead of the search some of the inferences which would be achieved during it by learning clauses. CDCL solvers can also benefit from 3-Resolution preprocessor—dramatically in certain instances—but the effects are far from uniform.

6.1 Future work

The following lines of research are open:

1. It would, of course, be easy if tedious to extend the experiments to more problem sets, more preprocessors and especially to more solvers. We shall probably look at some more DPLL solvers, but do not expect the results to add much more than detail to what is reported in the present paper. One of the more important additions to the class of solvers will be a non-clausal (Boolean circuit) reasoner. We have not yet experimented with such a solver. We have already investigated preprocessing for several state of the art SLS (stochastic local search) solvers, but that is such a different game that we regard it as a different experiment and do not report it here.

2. The more important line of research is to investigate methods for automatically choosing among the available preprocessors for a given problem instance, and

for automaticallly choosing the order in which to apply successive preprocessors. Machine learning may help here, though it would be better, or at least more insightful, to be able to base decisions on a decent theory about the interaction of reasoning methods.

3. Another interesting project is to combine preprocessors not as a series of separate modules but as a single reasoner. For example, it would be possible to saturate under 3-Resolution and hyper-resolution together, in the manner found in resolution-based theorem provers. Whether this would be cost-effective in terms of time, and whether the results would differ in any worthwhile way from those obtained by ordering separate preprocessors, are unknown at this stage.

As SAT solvers are increasingly applied to real-world problems, we expect deductive reasoning by preprocessors to become increasingly important to them.

Acknowledgments

This work was funded by National ICT Australia (NICTA). National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

[AMS03] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. Shatter: Efficient symmetry-breaking for boolean satisfiability. In *Design Automation Conference*, pages 836–839. ACM/IEEE, 2003.

[APSS05] Anbulagan, Duc Nghia Pham, John Slaney, and Abdul Sattar. Old resolution meets modern SLS. In *Proceedings of 20th AAAI*, pages 354–359, 2005.

[APSS06] Anbulagan, Duc Nghia Pham, John Slaney, and Abdul Sattar. Boosting SLS performance by incorporating resolution-based preprocessor. In *Proceedings of Third International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS), in conjunction with CP-06*, pages 43–57, 2006.

[AS05] Anbulagan and John Slaney. Lookahead saturation with restriction for SAT. In *Proceedings of 11th CP*, pages 727–731, 2005.

[ASM03] Fadi A. Aloul, Karem A. Sakallah, and Igor L. Markov. Efficient symmetry breaking for boolean satisfiability. In *Proceedings of 18th IJCAI*, Mexico, 2003.

[Bac02] Fahiem Bacchus. Enhancing Davis Putnam with extended binary clause reasoning. In *Proceedings of 18th AAAI*, pages 613–619, Edmonton, Canada, August 2002. AAAI Press.

[Bra01] Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proceedings of 17th IJCAI*, pages 515–522, 2001.

[Bra04] Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(1):52–59, 2004.

[BW04] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Revised Selected Papers of SAT 2003, LNCS 2919 Springer*, pages 341–355, 2004.

[DD01] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of 17th IJCAI*, pages 248–253, Seattle, Washington, USA, 2001.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of 8th SAT, LNCS Springer*, 2005.

[ES03] Niklas Eén and Niklas Sorensson. An extensible SAT-solver. In *Proceedings of 6th SAT*, 2003.

[HvM04] Marijn Heule and Hans van Maaren. Aligning CNF- and equivalence-reasoning. In *Proceedings of 7th SAT*, Vancouver, Canada, 2004.

[HvM06] Marijn Heule and Hans van Maaren. March_dl: Adding adaptive heuristics and a new branching strategy. *Journal on Satisfiability, Boolean Modeling and Computation*, (2):47–59, 2006.

[LA97] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of 3rd CP*, pages 341–355, Schloss Hagenberg, Austria, 1997.

[Li00] Chu Min Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proceedings of 17th AAAI*, pages 291–296, USA, 2000. AAAI Press.

[LMS01] I. Lynce and J. Marques-Silva. The interaction between simplification and search in propositional satisfiability. In *Proceedings of CP'01 Workshop on Modeling and Problem Formulation*, 2001.

[MMZ+01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC*, pages 530–535, 2001.

[OGMS02] Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from CNF formulas. In *Proceedings of 8th CP*, pages 185–199, 2002.

[Qui55] W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, 1955.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Rya04] Lawrence O. Ryan. *Efficient Algorithms for Clause Learning SAT Solvers*. PhD thesis, Simon Fraser University, Burnaby, Canada, 2004.

[SE05] Niklas Sorensson and Niklas Eén. MINISAT v1.13 - A SAT solver with conflict-clause minimization. In *2005 International SAT Competition website: http://www.lri.fr/∼simon/contest05/results/descriptions/solvers/minisat_static.pdf*, 2005.

[SKM97] Bart Selman, Henry Kautz, and David McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of 15th IJCAI*, pages 50–54, Nagoya, Aichi, Japan, 1997.

[SP05] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. In *Revised Selected Papers of SAT 2004, LNCS 3542 Springer*, pages 276–291, 2005.

[WvM98] Joost P. Warners and Hans van Maaren. A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23:81–88, 1998.

[ZMMM01] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of International Conference on Computer Aided Design ICCAD2001*, 2001.