

# Boosting SLS Performance by Incorporating Resolution-based Preprocessor

Anbulagan<sup>1</sup>, Duc Nghia Pham<sup>2</sup>, John Slaney<sup>1,3</sup>, Abdul Sattar<sup>2,4</sup>

<sup>1</sup>Logic and Computation Program, National ICT Australia Ltd.

<sup>2</sup>Safeguarding Australia Program, National ICT Australia Ltd.

<sup>3</sup>Computer Sciences Laboratory, Australian National University

<sup>4</sup>IIS, Griffith University, QLD, Australia

{anbulagan, duc-nghia.pham, john.slaney, abdul.sattar}@nicta.com.au

**Abstract.** State of the art Stochastic Local Search (SLS) solvers have difficulty in solving many CNF-encoded realistic SAT problems, apparently because they are unable to exploit hidden structure as well as systematic solvers. Recent work has shown that SLS solvers may benefit from a preprocessing phase borrowed from systematic SAT solving. In this paper, we report an extensive empirical examination of the impact of SAT preprocessing on the performance of contemporary SLS solvers. It emerges that all the examined solvers do indeed benefit from preprocessing, and the effect of each preprocessor is close to uniform across solvers and across problems. Our results suggest that SLS solvers need to be equipped with multiple preprocessors if they are ever to match the performance of systematic solvers on highly structured problems.

## 1 Introduction

When stochastic local search (SLS) began to be seriously applied to propositional satisfiability problems, in the early 1990s, it seemed promising as the technique of choice wherever proofs of unsatisfiability or of strict optimality are not required. SLS brought within range problems of previously thought inaccessible, solving randomly generated problems with thousands of variables. Unfortunately, despite many improvements in the algorithms, SLS solvers have generally failed to realise that early promise. For satisfiable random clause sets, they are indeed far more effective than any systematic reasoners, but on more realistic problems taken from practical applications they remain disappointing. By contrast, systematic solvers have improved progressively. Modern DPLL solvers, based on clause learning or lookahead, can solve many highly structured problems with millions of clauses, in bounded model checking, for instance. The results of the International SAT competitions (<http://www.satcompetition.org/>) are instructive: while SLS solvers occupied all the leading positions in the section on satisfiable random problems, the best solvers for (satisfiable) hand-crafted and industrial problems were all clause learning DPLL variants.

In outline, the explanation is clear: clause sets derived from real-world problems exhibit a great deal of structure such as symmetries, variable dependencies, clustering and the like. This enhances the effect of deductive reasoning at

each node of the search tree by giving unit propagation more opportunities to “bite”, while at the same time yielding detectable differences between clauses and between variables that assist the heuristics for choosing variables on which to branch. SLS solvers have no way to exploit this rich structure, because they have no propagation mechanisms and so do little or no reasoning.

Combining systematic and local search, so that each may help the other, is a natural idea but hard to achieve: embedding either style of search in the other tends to incur prohibitive overheads. Consequently, there has been recent interest in the alternative of reasoning offline, in a preprocessing phase, applying the SLS solver to the problem only after it has been refined, reduced or otherwise probed by some form of inference. Not all of the exploitable structure is detected by such a pre-search procedure, but the effects may nonetheless be enough to bring an important range of industrially relevant problems within the scope of SLS.

Our departure point for the present paper is the work reported by Anbulagan *et al.* [1] who enhanced a number of contemporary SLS solvers with a resolution-based preprocessor. One of the resulting solvers, R+AdaptNovelty<sup>+</sup>, won the random SAT category of the 2005 International SAT (SAT2005) competition, though it is still not competitive with the best DPLL solvers on more highly structured problems. The experiments reported by Anbulagan *et al.* are limited to a fairly small problem set and just examine the 3-Resolution preprocessor. To date, many powerful and complex preprocessors have been developed and have become a necessary element in the success of many systematic SAT solvers. However, there has been no systematic investigation that evaluate the effects of these preprocessors on SAT solvers, especially on local search techniques. In the present paper, we attempt to address this lack. We report the results of an extensive empirical evaluation of several preprocessors, mainly from recent, high-performance systematic solvers, and four SLS solvers which can reasonably claim to represent the state of the art. The problems used in the study are known to be hard for SLS.

Briefly, it emerges that while all the examined solvers do indeed benefit from preprocessing, the effect of each preprocessor is close to uniform across solvers and across problems. On most problems with realistic structure, the right choice of preprocessor, combined with the right choice of solver, results in performance superior to any previously achieved. What is the right choice, however, requires an extensive empirical study to answer. For any of the solvers, the wrong preprocessor can make things worse. There is an interplay between problem structure (itself a poorly understood notion), preprocessor inference mechanism and SLS method.

## 1.1 Related work

The most directly related work is that of Anbulagan *et al.* [1], as noted above. They only examine the impact of 3-Resolution preprocessor on the performance of SLS solvers, however. During the last decade, many other preprocessors have been applied to propositional reasoners. Among them are 2-SIMPLIFY [2],

2cl [3], the preprocessor in LSAT [4] for recovering and exploiting Boolean gates, a preprocessor based on probing [5], HyPre [6, 7], Shatter [8] for dealing with symmetry structure, NiVER [9] and SatELite [10]. We consider some of these preprocessors plus 3-Resolution in our experiments. Interesting recent lines of work on stochastic/systematic hybrids include the Complete Local Search technique [11, 12] which is however orthogonal to our research: it may be interesting to investigate how it interacts with preprocessors, but we have not yet done this.

## 1.2 Plan of the paper

The next section outlines the five different preprocessors we use for the experiments. We then briefly examine their effect on the sizes of the problems on which the solvers are to be run, and go on to describe the four SLS solvers to be compared. We then present the experimental results together with a careful analysis of the behaviours of four contemporary SLS solvers that are enhanced by different preprocessors across the benchmark set. We also propose a multiple preprocessing approach to further boost the performance of SLS SAT solvers, before we conclude with some remarks and suggestions for future work.

## 2 The Preprocessors

SAT Preprocessors are introduced to exploit hidden structure in many hard SAT problems, whether these be small problems, such as par16\* and par32\*, or large industrial problems, such as those from bounded model checking. Preprocessing may either reduce or increase the size of the formula, as new clauses are deduced and old ones subsumed or otherwise removed. In the following subsections, we present the state of the art preprocessors used in our empirical study: 3-Resolution, 2-SIMPLIFY, HyPre, NiVER and SatELite.

### 2.1 3-Resolution

The systematic solver Satz [13] used a restricted resolution procedure, called 3-Resolution, to preprocess input formulas before running a complete backtrack search. This procedure computes resolvents for all pairs of clauses of length  $\leq 3$ . If the length of the inferred resolvent is also  $\leq 3$ , then it is added to the formula and in turns is used to produce other resolvents. The procedure is repeated until saturation. Duplicate and subsumed clauses are deleted, as are tautologies and any duplicate literals in a clause.

Recently, this approach helped R+AdaptNovelty<sup>+</sup> [1] win the satisfiable random problem category in the SAT2005 competition.

### 2.2 2-SIMPLIFY

Brafman [2] developed the 2-SIMPLIFY preprocessor to treat SAT instances with many binary clauses from classical AI planning problems. Given a formula  $\mathcal{F}$ , 2-SIMPLIFY constructs an implication graph from all binary clauses

in  $\mathcal{F}$ , and uses transitive reduction to deduce unit literals from this graph. Unit propagation are then applied to these deduced literals to further simplify and update the implication graph. 2-SIMPLIFY also utilises this graph to derive more *shared implications*, a restricted variant of hyper-resolution. The process is repeated until  $\mathcal{F}$  is stable. This preprocessor was further enhanced by implementing subsumption and pure literal deduction rules [14].

Empirical results [2, 14] show that while systematic search benefits markedly from 2-SIMPLIFY on all tested problems, the impact on the SLS solver WalkSAT varies according to the problem type.

### 2.3 HyPre

Like 2-SIMPLIFY, HyPre [7] reasons with binary clauses. However, by incorporating full hyper-resolution, HyPre is more general and powerful than 2-SIMPLIFY [7]. Initially, this principle was implemented as part of the development of a very competitive SAT solver, 2CLS+EQ [6]. HyPre uses hyper-resolution to infer new binary clauses and hence avoids the space explosion of computing a full transitive closure. In addition, unit and equality reductions are incrementally applied to infer more binary clauses, making the preprocessor still more effective.

### 2.4 NiVER

Another variant of resolution used in preprocessing is Variable Elimination Resolution (VER), first proposed as part of the well-known systematic Davis-Putnam (DP) procedure [15]. For each variable  $v$ , VER replaces all clauses containing  $v$  and  $\bar{v}$  with their resolvents. As a result, variable  $v$  is eliminated from the new formula. If repeated on other variables until all literals are pure or the empty clause is derived, this process can serve as a decision procedure.

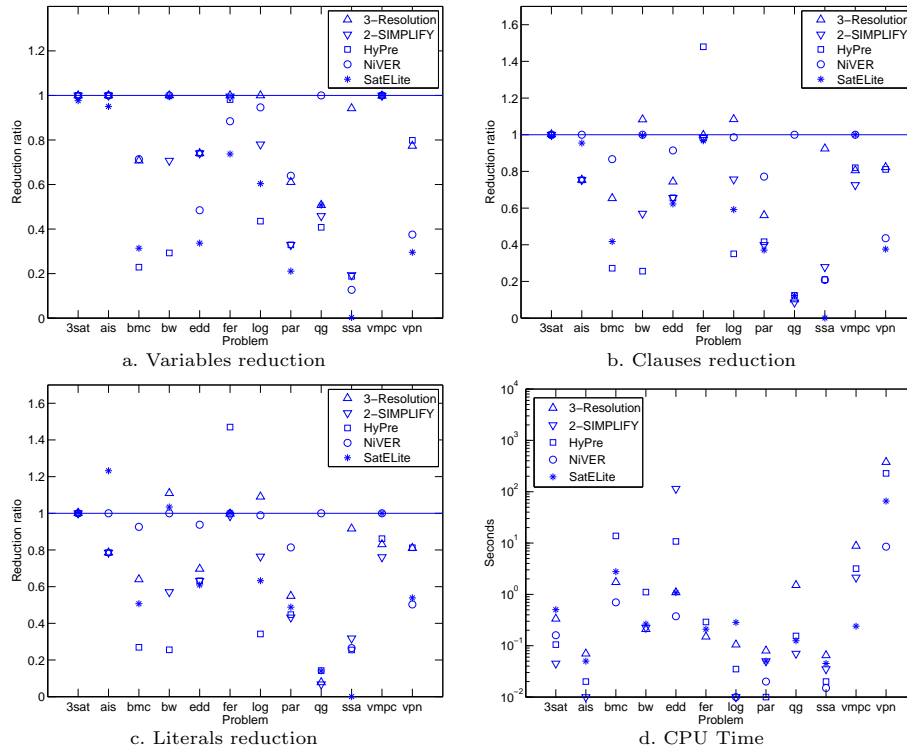
The application of VER to large problem instances is limited by its exponential space complexity. Recently, Subbarayan and Pradhan [9] observed that most resolvents deduced by VER on real-world instances are tautologies and hence redundant. Consequently, they proposed a linear space preprocessor version of VER, namely Non increasing VER (NiVER) by restricting the variable elimination to happen only if there is no increase in the number of literals after elimination. Experimental results showed that NiVER significantly improved the performance of systematic SAT solvers [9].

### 2.5 SatELite

Later, Eén and Biere [10] further improved NiVER in both reduction and speed performances. The resulting SatELite preprocessor extends NiVER with a Variable Elimination by Substitution rule. Deleting subsumed clauses boosts the performance of NiVER in space reduction by 30% [10]. The time performance of SatELite (and NiVER) was also improved with the use of *touched lists* and *clause*

*signatures*. As a result, SatELiteGTI (a combination between SatELite preprocessor and MiniSAT solver) dominated the SAT2005 competition on the crafted and industrial problem categories and won four of nine gold medals awarded in this competition.

### 3 Formula Reduction by Preprocessing



**Fig. 1.** The impact of preprocessors on SAT problems.

Figure 1 shows the impact of the 5 preprocessors on a range of SAT problems taken from SAT2005’s hard random 3-SAT (10 instances), quasigroup existence (10 instances) and ten realistic domains:

- 6 instances of all interval series (ais),
- 3 bounded model checking problems bmc-ibm\* (bmc),
- 4 instances of blocks-world planning (bw),
- 6 instances of job-shop scheduling e\*ddr\* (edd),
- 5 instances of ferry planning (fer) taken from SAT2005,

4 logistics planning problems (log),  
5 instances of parity problems par16\* (par),  
4 “single stuck-at” problems (ssa),  
5 cryptographic problems (vm<sub>pc</sub>) from SAT2005, and  
2 models generated from Alloy (vpn) from SAT2005.

Unless otherwise stated, the problem instances are taken from SATLIB and all are known to be hard for SLS. The smallest input problem contains 61 variables and 581 clauses (ais6) and the largest one contains 267,766 variables and 1,002,957 clauses (vpn-1962).

Figure 1 shows the median reductions in number of variables, number of clauses and number of literals, and median runtime, over each problem domain using each preprocessor. Reduction rate is defined as reduced size divided by initial size. We observe from Figures 1a, 1b and 1c that all of the preprocessors reduce the size, in variables, clauses or literals, of most of the problems. In the case of random 3-SAT, the effect is small, as expected where there is low impact of the simplification techniques incorporated in each preprocessor. No one preprocessor is best or worst across the whole range of problem classes, though for each problem class (except 3-SAT) some preprocessors are more effective than others. In a few anomalous cases, notably HyPre on the ferry planning problems, preprocessing appears to increase the problem size. We also observe from Figures 1b and 1c that SatELite increases the density of literal occurrences for a few problems, especially the ais problem. Overall, the most effective preprocessors appear to be HyPre, SatELite and 2-SIMPLIFY. This finding has to be set against the runtime cost—all three can be expensive in certain cases—and of course it is another question whether reductions in problem size translate into improvements in the behaviour of SLS solvers.

## 4 The SLS Solvers

Although Novelty [16], one of the best contemporary solvers in the WalkSAT family, can solve many hard problems better than systematic search, it may loop indefinitely and fail to return a solution due to its deterministic variable selection [17, 18].<sup>1</sup> Hoos [17] solved this problem by adding probabilistic random walks to Novelty, resulting in Novelty<sup>+</sup>. Later, Li and Huang [18] proposed a more diversified heuristic to weaken the determinism in Novelty: within a diversification probability, the recently least flipped variable is selected for the next move, otherwise the search performs as Novelty. They further improved this algorithm by weakening the randomness in the WalkSAT family. WalkSAT and its variants first randomly pick an unsatisfied clause and then select the next move from variables of this clause. Li and Huang [18] consider variables from

---

<sup>1</sup> Novelty deterministically selects the next move from the two best variables of a randomly selected false clause [16]. Hoos [17] gave an example of satisfiable instance on which Novelty loops indefinitely and is unable to find a solution regardless of the noise parameter setting.

all unsatisfied clauses and use a sophisticated gradient-based greedy heuristic to find the next promising move. The resulting solver g2wsat, that implements these two heuristics, finished second in the SAT2005 competition for random SAT category.

Another of the best solvers in the WalkSAT family is AdaptNovelty<sup>+</sup> [19], an automated version of Novelty<sup>+</sup> that won the random SAT category of the SAT2004 competition. As with other WalkSAT variants, the performance of Novelty<sup>+</sup> critically depends on the setting of its noise parameter, which controls the greediness of the search. AdaptNovelty<sup>+</sup> addresses this problem by adaptively tuning its noise level based on the detection of stagnation. Experimental results have shown that this adaptive noise mechanism also works well with other WalkSAT variants [19].

Recently, clause weighting based SLS algorithms, most notably PAWS [20] and SAPS [21], have been very successfully applied to hard combinatorial SAT problems. These algorithms dynamically update the clause weights (or penalties) and hence modify the search landscape to effectively avoid or escape local minima during the search. The underlying weighting strategy of these solvers is based on two mechanisms: *increasing* (or *scaling*) and *reducing* (or *smoothing*). When the search encounters a local minimum, it increases the weights of current unsatisfied clauses. As violating these unsatisfied clauses now costs more than violating other satisfied clauses, the search is forced to move to a new neighbour to satisfy those current unsatisfied clauses. As a result, it escapes the local minimum. After a number of weight increases, the weights of all weighted clauses are reduced to forget about the high costs of violating clauses which are no longer helpful since it escaped the local minima. These two mechanisms significantly increase the mobility of the search as well as helping to focus it on any “critical” clauses.

## 5 Experimental Results

As noted above, we investigated the effect of preprocessing on SLS for 64 problem instances drawn from 12 classes of problems. Each of the five preprocessors was applied to each problem instance, resulting in 320 reduced problem instances. Each of these was solved 100 times with different random number seeds by each of the four SLS solvers, with a time limit of 1200 seconds. This gives a data set consisting of 100 sample points for each of 1280 triples of ⟨problem, preprocessor, solver⟩. In addition, we also run the 4 SLS solvers on the original problems. The experiment involved 153,600 runs on a Linux Pentium IV computer with 3.0GHz CPU and 1GB RAM.

Table 1 summarises by giving success rates (percentage of runs resulting in a solution) and runtimes just for the one or two hardest instances in nine of the twelve problem sets.<sup>2</sup> The “hardest” instance is defined to be the one giving the lowest aggregate success rate for all four solvers, unaided by preprocessing. The runtimes are means, with a maximum in each case of 600 seconds, except on the

---

<sup>2</sup> PAWS<sub>10</sub> is a variant of PAWS with the smooth parameter fixed to 10. RSAPS is the reactive version of SAPS.

Instances	Prep.	#Vars/#Cls/#Lits	Ptime	g2wsat		AdaptNovelty <sup>+</sup>		PAWS <sub>10</sub>		RSAPS	
				%	Stime	%	Stime	%	Stime	%	Stime
ais14	origin	365/6969/16615	n/a	75	295.59	55	424.33	100	27.64	100	4.68
	2-SIM	365/6969/16615	0.04	69	304.75	68	373.03	100	19.26	100	4.34
	HyPre	365/6969/16615	0.03	67	338.46	48	427.13	100	23.58	100	2.15
	SatELite	351/6773/22257	0.09	100	11.28	100	4.73	6	564.03	100	3.20
ais16	origin	481/10621/25261	n/a	1	596.01	2	591.38	75	329.96	100	48.11
	2-SIM	481/10621/25261	0.06	10	558.64	3	593.65	66	352.13	100	25.72
	HyPre	481/10621/25261	0.04	5	583.78	1	596.47	66	362.62	100	43.77
	SatELite	465/10365/34140	0.17	96	147.93	100	50.42	2	588.07	99	79.73
bw_large_c	origin	3016/50457/114314	n/a	30	511.20	100	24.07	100	57.67	100	42.67
	3-Res	3016/54563/126629	0.30	20	547.51	100	50.30	100	63.56	100	68.85
	2-SIM	2176/29770/67497	0.48	1	597.57	91	237.01	100	13.83	100	7.17
	HyPre	1200/16446/37316	2.05	100	6.54	100	18.75	100	0.58	100	0.23
SatELite	3002/50264/118175	0.47	15	558.70	100	27.18	100	62.10	100	53.45	
bw_large_d	origin	6325/131973/294118	n/a	0	600.00	99	165.36	24	518.42	45	460.53
	3-Res	6325/141421/322458	0.85	0	600.00	88	244.74	15	551.18	34	485.78
	2-SIM	4644/82453/183302	1.57	0	600.00	37	481.55	98	149.17	99	139.24
	HyPre	3572/86285/188327	15.08	51	410.81	75	338.64	100	21.36	100	41.97
SatELite	6307/131653/303116	1.22	0	600.00	99	123.38	25	525.46	35	468.96	
e0ddr2-5-1	origin	17490/103887/286193	n/a	100	50.60	100	174.05	17	523.77	100	2.15
	3-Res	13539/81864/211802	1.08	100	47.92	100	22.92	22	472.99	100	1.36
	2-SIM	13539/72850/193774	98.00	94	175.91	99	168.16	33	413.88	100	1.59
	HyPre	13539/72925/193924	7.81	94	185.57	100	162.00	32	422.14	100	1.73
NiVER	8448/94845/268109	0.78	98	53.79	0	600.00	16	519.70	100	1.24	
SatELite	6209/68569/185212	1.06	100	2.07	37	495.11	41	364.64	100	0.57	
ferry8-ks9a-4004	origin	1259/15259/31167	n/a	58	508.44	2	1197	86	543.26	100	0.08
	3-Res	1241/15206/31071	0.11	64	435.55	12	1150	98	285.26	100	0.04
	2-SIM	1233/14562/29783	0.00	56	531.99	26	1046	94	431.19	100	0.07
	HyPre	1209/20906/42471	0.13	30	947.43	8	1136	100	1.34	100	0.04
NiVER	976/14952/30817	0.00	58	510.05	40	937.56	100	28.14	100	0.03	
SatELite	813/14720/34687	0.19	84	210.65	100	209.11	100	4.14	100	0.03	
log_d	origin	4713/21991/51347	n/a	100	8.85	100	0.30	100	1.17	100	0.17
	2-SIM	3988/16602/39687	0.60	100	0.16	100	0.15	100	0.09	100	0.06
	HyPre	3834/25605/57434	1.23	100	0.13	100	0.11	100	0.06	100	0.05
	NiVER	4430/21450/50922	0.14	100	3.79	100	0.49	100	0.29	100	0.09
SatELite	3032/18309/51479	0.61	100	3.61	100	0.51	100	0.24	100	0.07	
par16-1	origin	1015/3310/8788	n/a	12	534.30	35	479.82	0	600.00	5	588.42
	3-Res	607/1815/4713	0.04	9	548.27	99	145.44	0	600.00	97	88.02
	2-SIM	317/1266/3652	0.03	74	312.23	100	54.26	10	567.43	100	27.92
	HyPre	317/1324/3790	0.01	63	331.39	100	32.47	8	566.36	99	18.37
NiVER	632/2512/7058	0.02	81	200.98	97	153.91	3	587.52	19	530.89	
SatELite	201/1173/4121	0.05	100	22.99	100	17.59	93	214.34	100	5.91	
par16-2	origin	1015/3374/9044	n/a	7	558.44	10	568.89	0	600.00	0	600.00
	3-Res	632/1929/5069	0.04	7	558.39	61	383.73	0	600.00	68	279.05
	2-SIM	349/1394/4036	0.09	32	479.43	79	329.16	4	589.93	92	101.95
	HyPre	349/1452/4174	0.01	48	458.59	90	214.40	0	600.00	96	52.58
NiVER	664/2640/7442	0.02	61	369.85	54	404.92	2	591.87	4	589.88	
SatELite	223/1301/4585	0.05	99	98.57	100	90.62	9	563.50	100	11.64	
qg5-11	origin	1331/64054/174879	n/a	98	37.21	0	600.00	8	580.89	100	73.08
	3-Res	824/27415/71672	1.23	98	38.95	99	72.03	100	1.19	100	2.55
	2-SIM	724/26024/68019	0.43	100	2.57	100	7.87	100	0.44	100	0.78
	HyPre	692/24743/64870	0.41	100	1.81	100	5.57	100	0.24	100	0.55
SatELite	824/28488/73818	0.22	100	41.50	97	82.26	100	1.91	100	2.56	
qg7-13	origin	2197/97072/256426	n/a	21	518.52	0	600.00	0	600.00	3	587.76
	3-Res	1412/45362/115164	2.06	27	511.52	49	381.89	99	106.34	99	159.64
	2-SIM	1333/41647/106430	0.75	75	326.24	84	135.23	100	37.97	100	58.96
	HyPre	1207/41110/105189	0.95	100	24.57	99	27.32	100	8.23	100	19.69
SatELite	1412/45967/116374	0.35	17	541.08	55	339.55	98	157.78	100	130.90	
vmc-1925	origin	784/108080/283220	n/a	14	1115	8	1145	4	1158	2	1197
	3-Res	784/84763/227748	5.75	0	1200	0	1200	0	1200	0	1200
	2-SIM	784/77745/213559	1.46	10	1111	8	1124	26	1044	10	1139
	HyPre	783/88835/244482	1.44	16	1112	16	1110	12	1115	8	1167
3sat-1648	origin	10000/42000/126000	n/a	68	744.99	16	1145	78	399.52	0	1200
	3-Res	10000/42081/126243	0.40	66	747.21	44	1022	80	477.08	0	1200
	NiVER	9982/41981/125993	0.43	78	600.19	30	1082	90	287.00	0	1200
	SatELite	9775/41747/126601	0.66	70	672.08	22	1133	74	463.08	0	1200
3sat-1656	origin	12000/50400/151200	n/a	12	1120	2	1196	50	729.51	0	1200
	3-Res	12000/50492/151476	0.52	24	1046	4	1196	76	538.16	0	1200
	NiVER	11971/50371/151188	0.53	10	1148	2	1195	68	567.44	0	1200
	SatELite	11698/50067/152106	0.82	12	1061	2	1199	64	575.35	0	1200

Table 1. Results of resolution enhanced SLS solvers.

very hard ferry, vmc and random problems, for which it is 1200 seconds. In this table, Ptime represents the preprocessing time, while Stime represents the runtime of each solver without Ptime. The preprocessors which give no impact on problem size have been omitted in the interests of shortening the table. No results are given in this table for the bmc, ssa or vpn domains because all the

instances in those domains are either so easy that the results are uninteresting or so hard (even after preprocessing) that again the results are uninteresting.

The all interval series problem of order  $n$  is to find a permutation of the numbers  $0, \dots, n - 1$  such that the intervals between neighbours in the series are also a permutation of  $1, \dots, n - 1$ . Problems `ais14` and `ais16`, which are the instances where  $n = 14$  and  $n = 16$  respectively, are reliably solved by RSAPS, hard for PAWS<sub>10</sub> and very hard for `g2wsat` and `AdaptNovelty+`. `SatELite` has a dramatic effect on the performance of the latter two solvers, though it is the worst of the preprocessors for the former two. Although it seems that `2-SIMPLIFY` and `HyPre` made no change to these instances, they indeed alternated the underlying structures of the two `ais` instances, resulting in the best improvement for PAWS<sub>10</sub> and RSAPS.

The “`bw_large`” problems are from a blocks-world planning domain. The largest, `bw_large_d`, is challenging for most solvers. `AdaptNovelty+` is an exception, finding solutions on 99% of runs even without preprocessing. Indeed, preprocessing, except by `SatELite`, actually slows it down. `g2wsat` is unable to solve this problem without preprocessing; with `HyPre` it succeeds on about half of the runs. The solvers PAWS<sub>10</sub> and RSAPS benefit greatly from `HyPre` and to a lesser extent from `2-SIMPLIFY`. All solvers benefit from `HyPre` on `bw_large.c` problem. In comparison to the other preprocessors, `HyPre` takes more time in simplifying the problems and yields the smallest refined ones.

The “`e*ddr*`” problems are from a job-shop scheduling domain, and all solvers except PAWS<sub>10</sub> can solve them reliably without preprocessing. The solvers `g2wsat`, PAWS<sub>10</sub> and RSAPS benefit more from `SatELite` than from the other preprocessors, but `AdaptNovelty+` does best with `3-Resolution`; `SatELite` actually makes it worse, though not as badly as `NiVER`. RSAPS for some reason appears well adapted to the structure in this problem. Enhanced with `SatELite`, it finds solutions on average in half a second.

“`Ferry`” is another planning domain, which challenges all the solvers except RSAPS, which again does remarkable well with it. `NiVER+RSAPS` has the best performance on this problem. Preprocessing helps somewhat, `SatELite` in particular succeeding in teasing out useful structure, and raises the success rate of `AdaptNovelty+` from 2% to 100%.

The logistics problems are too easy to stress any of these four SLS solvers, but were included because they show that even where performance is already good, SLS solvers can benefit from preprocessing. `SatELite` takes significant time to preprocess the problems. `log_d` is the hardest of them. `HyPre`, `2-SIMPLIFY` and `SatELite` are generally effective in improving runtime of the solvers on this problem, without considering their preprocessing times.

The parity learning problems “`par16*`” are all hard for SLS solvers, though some systematic solvers, especially those specialised for equality reasoning find them comparatively easy. We give data on two instances in the table rather than one, because these problems demonstrate the effects of preprocessing particularly well. `SatELite` exploits the problem structure to notable effect, raising performance in most cases from near zero to 100% or almost 100%. The figures

for the other preprocessors do not show such a clear pattern, but all are effective in at least some cases.

The quasigroup existence problems, of which “QG5.11” and “QG7.13” are instances, have sometimes been thought unsuitable for SLS, and indeed early solvers such as GSAT found them inaccessible. Recent SLS solvers, however, have more success with them. The main effect of preprocessing is to implement unit propagation, which reduces the problem size markedly. NiVER fails to do this, and so has no effect, but all other preprocessors help significantly. HyPre is a clear winner here, though 2-SIMPLIFY is also helpful. PAWS<sub>10</sub> with HyPre achieves a performance which compares well with that of contemporary systematic solvers. Without preprocessing it cannot find solutions at all on “QG7.13” instance.

The problem “vmc-1925” comes from VLSI design and is genuinely hard for all four solvers, with or without preprocessing. The highest success rate achieved is 26% by PAWS<sub>10</sub> with 2-SIMPLIFY. 3-Resolution, for reasons not yet completely clear, makes the problem even harder for all solvers. HyPre has some beneficial effect, though it is not dazzling.

Finally, although our focus is on structured problems, we find it interesting that positive effects are found in the case of purely random problems as well, though as we might expect they are not as impressive as those obtained in some other cases. 3-Resolution reduces random problems significantly, and the variable elimination in NiVER also helps. HyPre and 2-SIMPLIFY do essentially nothing on random 3-SAT problems. The problem instances here have 10,000 and 12,000 variables respectively and come from the “hard” region with a ratio of clauses to variables of 4.2.

## 6 Analysis

### 6.1 Matching Preprocessors to Solver-Problem Pairs

Solvers\Problems	ais	bw	e*ddr	ferry	log	par16	qg	ssa	vmc	3sat
AdaptNovelty <sup>+</sup>	Sat	Sat	3-Res	Sat	HyP	Sat	HyP	Sat	HyP	NiV
g2wsat	Sat	HyP	Sat	Sat	HyP	Sat	HyP	Sat	Sat	Sat
PAWS <sub>10</sub>	2-SIM	HyP	Sat	HyP	HyP	Sat	HyP	Sat	2-SIM	3-Res
RSAPS	HyP	HyP	Sat	HyP	HyP	Sat	HyP	Sat	2-SIM	n/a

**Table 2.** The best preprocessor for SLS-problem pair.

Based on the results, we identify and summarise in Table 2 the preprocessor that provide the most average improvement for each examined solver on each benchmark problem domain. While there is no absolute “winner” among the preprocessors, it is clear that for most of the problem classes we examined, and for most of the solvers, SatELite is a good choice. In details, SatELite is the favourite choice of all four solvers for the parity (par16), “single stuck-at” (ssa) problems and of non-weighting solvers for the all interval series (ais), ferry planning problems. This is perhaps not too surprising given that it is the

most complex of these preprocessors. However, due to its novel implementation, SatELite records competitive runtimes in comparison with other preprocessors.

On the other hand, HyPre is also valuable in most cases. In fact, it is the preferable choice of all four solvers to exploit the structures of the planning (bw, ferry and log), quasigroup existence (qg) problems.

In addition, it is worth noting that there is no uniform winner among these preprocessors for the random 3-SAT problems. Indeed, each of the three solvers prefers a different preprocessor.<sup>3</sup> However, the best results for these random instances were achieved from 3-Resolution+PAWS<sub>10</sub>.

## 6.2 Matching Preprocessors to Solvers

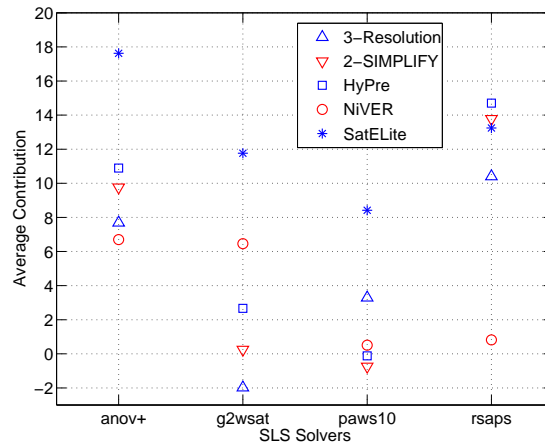


Fig. 2. Average contribution of preprocessor to SLS based on success rate.

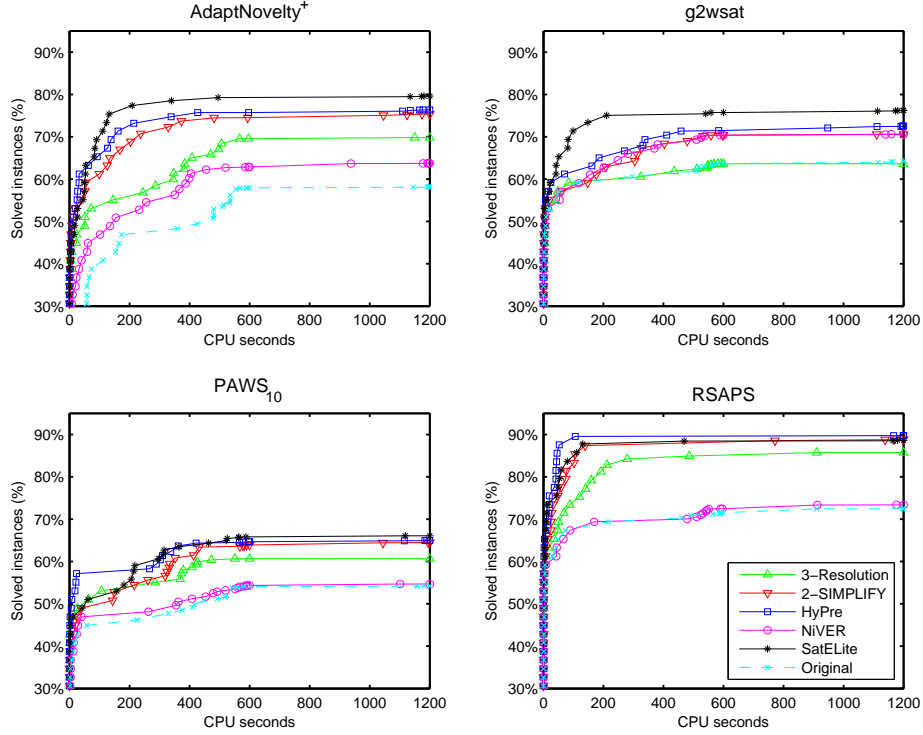
Figure 2 shows the average contribution in percentage terms of each preprocessor to each SLS solver. The average contribution of a preprocessor to a SLS solver is computed based on all problems used in our experiment, excluding bmc and vpn, and defined as the difference between the solver’s overall (percentage) success rate after preprocessing and its success rate before preprocessing.

The heuristics used in AdaptNovelty<sup>+</sup>, g2wsat and PAWS<sub>10</sub> are most compatible on this measure with SatELite preprocessor, while RSAPS is more compatible with HyPre. We re-emphasise, however, that this is an overall measure, and the effects vary considerably from one problem class to another as shown in Table 2.

## 6.3 Runtime Distributions

To further illustrate the significant improvement on the performance of SLS solvers when each preprocessor is performed, we graph the runtime distributions

<sup>3</sup> RSAPS totally failed to solve all random instances with or without preprocessing.



**Fig. 3.** Runtime performance of resolution-enhanced SLS solvers.

(RTDs) of these solvers over all structured problems except `bmc` and `vpn`, in Figure 3. Among four SLS solvers examined, `AdaptNovelty+` benefits mostly from all five resolution-based preprocessors, as with an aggregate 22% boost from the original successful rate. In addition, the impacts of these simplifiers on `AdaptNovelty+` are clearly different and separated from each others. For other three solvers, those preprocessing impacts are lesser distinguished as the RTDs of their preprocessor+solver variants are greatly crossed over and overlapped. Indeed, `PAWS10` achieves the smallest general enhancement when it is combined with preprocessors, followed by `g2wsat` and `RSAPS`.

Although `AdaptNovelty+` receives the greatest improvement from preprocessors, it is totally outperformed by `RSAPS` across the whole set of benchmark problems, except for the random 3-SAT problem. This observation is also applied to the cases of `g2wsat` and `PAWS10`. It indicates that the heuristic implemented in `RSAPS` is somehow better adapted to the structures of the realistic benchmark problems than other solvers. As shown in Figure 3, `RSAPS` without preprocessing achieves 73% successful rate and even outperformed all `PAWS10` variants and most variants of the other two solvers. When enhancing it by preprocessors, all `RSAPS` variants except `NiVER+RSAPS`, dominate all other solver variants

whereas HyPre+RSAPS is the best winner with an average record of 90% successful rate.

#### 6.4 Clause Weighting versus Random Walk

As shown in Figure 3, the magnitude of improvement of solvers varies dramatically depending on heuristics employed in the solvers. For WalkSAT family solvers, e.g. AdaptNovelty<sup>+</sup> and g2wsat, SatELite is the promising choice and clearly outperforms the second option, HyPre. However, this pattern is reversed when applied to the clause weighting solvers PAWS<sub>10</sub> and RSAPS. The RTD of HyPre closely catches up and overlaps with the one of SatELite in the case of PAWS<sub>10</sub>, and becomes dominant when applied to RSAPS. In addition, the improvement provided by 2-SIMPLIFY to RSAPS are also very competitive to SatELite.

Among five examined preprocessors, while the performance of SLS solvers are boosted significantly by SatELite, HyPre and 2-SIMPLIFY, the contribution of 3-Resolution and NiVER to SLS are smaller and lesser reliable. The impact of NiVER is hardly noticeable from the original results on g2wsat, PAWS<sub>10</sub> and RSAPS. The poor performance of NiVER is because it provides the least simplification when preprocessing the structured problem instances in our study.

### 7 Multiple Preprocessing and Preprocessor Ordering

Instances	Preprocessor	#Vars/#Cls/#Lits	Ptime	Succ. rate	CPU Time		Flips	
					median	mean	median	mean
ferry7-ks99i-4001	origin	1946/22336/45706	n/a	100	192.92	215.27	55,877,724	63,887,162
	SatELite	1286/21601/50644	0.27	100	4.39	5.66	897,165	1,149,616
	HyPre	1881/32855/66732	0.19	100	2.34	3.26	494,122	684,276
	HyPre & Sat	1289/29078/76551	0.72	100	2.17	3.05	359,981	499,964
	Sat & HyPre	1272/61574/130202	0.59	100	0.83	1.17	83,224	114,180
ferry8-ks99i-4005	origin	2547/32525/66425	n/a	42	1,200.00	910.38	302,651,507	229,727,514
	SatELite	1696/31589/74007	0.41	100	44.96	58.65	7,563,160	9,812,123
	HyPre	2473/48120/97601	0.29	100	9.50	19.61	1,629,417	3,401,913
	HyPre & Sat	1700/43296/116045	1.05	100	5.19	10.86	1,077,364	2,264,998
	Sat & HyPre	1680/92321/194966	0.90	100	2.23	3.62	252,778	407,258
par16-4	origin	1015/3324/8844	n/a	4	600.00	587.27	273,700,514	256,388,273
	HyPre	324/1352/3874	0.01	100	10.14	13.42	5,230,084	6,833,312
	SatELite	210/1201/4189	0.05	100	5.25	7.33	2,230,524	3,153,928
	Sat & HyPre	210/1210/4207	0.05	100	4.73	6.29	1,987,638	2,655,296
	HyPre & Sat	198/1232/4352	0.04	100	1.86	2.80	1,333,372	1,995,865

**Table 3.** RSAPS performance on ferry planning and par16-4 instances.

The preprocessors in our study sometimes show quite different behaviour on the same problem. One may increase the size of a given formula, while another decreases the number of clauses or number of variables or number of literals. It therefore seems reasonable to consider running multiple preprocessors on a hard formula before solving the preprocessed formula using a SLS solver. Table 3 shows preliminary results from an experiment with just three cases. The solver used in this experiment was RSAPS, and the preprocessors SatELite and HyPre. The selected problems are from ferry planning and par16 domains. We compare

the effects of running each preprocessor separately, then of running SatELite after HyPre, and finally of running SatELite followed by HyPre.

We observe from Table 3 that running SatELite followed by HyPre, on ferry planning instances, is the best. However, the order of running these preprocessors is reversed when applied to par16-4 instance. These preliminary results show that the performance of a SLS solver such as RSAPS can be improved orders of magnitude, using multiple preprocessors and by selecting the right order of preprocessors. However, what is the right combination as well as what is the right order of preprocessors remain unclear to us. This opens the way to a yet more complex study of preprocessor combinations and their use with different SLS and systematic solvers.

## 8 Conclusion

The aim of the study was to examine and analyse the impact of state of the art resolution-based SAT preprocessors on the performance of contemporary SLS solvers for satisfiability. Starting from reports in the recent literature of the usefulness of preprocessing in enhancing the performance of SLS solvers, we have conducted the first extensive and systematic empirical study of this issue with respect to a range of preprocessors and state of the art SLS solvers. We hoped and expected that preprocessing would bring clear benefits in performance on highly structured problems, and found this indeed to be the case for a fair range of problems, though not for all. In addition, many highly structured problems (such as, blocks-world planning, ferry planning, parity learning and quasigroup existence), which are usually thought unsuitable for local search, have been brought within the scope of certain SLS solvers using this approach.

The outcome of this study is that the “best” preprocessor and solver combination is highly problem-dependent, although SatELite or HyPre do pretty well with all the solvers in different problem domains. Certainly the benefit from preprocessing is *not* due simply to reduction in formula size, for the effect of the reduced formula is not uniform across SLS solvers, which depend on clause weighting or on random walk heuristics. Nor is there one type of preprocessor inference mechanism, apart from very simple ones such as unit propagation, that is best across problems or across solvers.

The most promising line of future research is to continue investigating combinations of preprocessors, as it seems that the ultimate goal of having SLS solvers exploit structure as systematic ones do may require different reasoners to supplement each other in this way.

## 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

1. Anbulagan, Pham, D.N., Slaney, J., Sattar, A.: Old resolution meets modern SLS. In: Proceedings of 20th AAAI. (2005) 354–359
2. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. In: Proceedings of 17th IJCAI. (2001) 515–522
3. Van Gelder, A.: Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. In: AMAI. (2002)
4. Ostrowski, R., Grégoire, E., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Proceedings of 7th CP. (2002) 341–355
5. Lynce, L., Marques-Silva, J.: Probing-based preprocessing techniques for propositional satisfiability. In: Proceedings of 15th ICTAI. (2003)
6. Bacchus, F.: Enhancing Davis Putnam with extended binary clause reasoning. In: Proceedings of 18th AAAI. (2002) 613–619
7. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: Revised Selected Papers of SAT 2003, LNCS 2919 Springer. (2004) 341–355
8. Aloul, F.A., Markov, I.L., Sakallah, K.A.: Shatter: Efficient symmetry-breaking for boolean satisfiability. In: Proceedings of DAC. (2003) 836–839
9. Subbarayan, S., Pradhan, D.K.: NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. In: Revised Selected Papers of SAT 2004, LNCS 3542 Springer. (2005) 276–291
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proceedings of 8th SAT, LNCS Springer. (2005)
11. Fang, H., Ruml, W.: Complete local search for propositional satisfiability. In: Proceedings of 19th AAAI. (2004) 161–166
12. Shen, H., Zhang, H.: Another complete local search method for SAT. In: Proceedings of LPAR. (2005) 595–605
13. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Proceedings of 3rd CP. (1997) 341–355
14. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **34** (2004) 52–59
15. Davis, M., Putnam, H.: A computing procedure for quantification theory. *Journal of the ACM* **7** (1960) 201–215
16. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for invariants in local search. In: Proceedings of 14th AAAI. (1997) 321–326
17. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. In: Proceedings of 16th AAAI. (1999) 661–666
18. Li, C.M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. In: Proceedings of 8th SAT, LNCS Springer. (2005)
19. Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In: Proceedings of AAAI-2002. (2002) 655–660
20. Thornton, J., Pham, D.N., Bain, S., Ferreira Jr., V.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of 19th AAAI. (2004) 191–196
21. Hutter, F., Tompkins, D.A.D., Hoos, H.H.: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proceedings of 8th CP. (2002) 233–248