

Trends in Constraint Programming

Frédéric BENHAMOU
Narendra JUSSIEN
Barry O'SULLIVAN

November 24, 2006

Contents

FIRST PART. LOCAL SEARCH TECHNIQUES IN CONSTRAINT SATISFACTION	11
Chapter 1. Local search techniques in Constraint satisfaction	13
Yehuda NAVEH , Andrea ROLI	
Chapter 2. Boosting SLS using Resolution	15
ANBULAGAN, Duc Nghia PHAM, John SLANEY , Abdul SATTAR	
2.1. The SLS Solvers	16
2.2. The Preprocessors	17
2.3. Empirical Evaluation	17
2.3.1. Clause Weighting versus Random Walk	18
2.3.2. Matching Preprocessors to Solver-Problem Pairs	18
2.3.3. Multiple Preprocessing and Preprocessor Ordering	19
2.4. Conclusion	20
2.5. Bibliography	20

FIRST PART

Local search techniques in Constraint
satisfaction

Chapter 1

Local search techniques in Constraint satisfaction

Here is the content of the chapter.

Chapter written by Yehuda NAVEH and Andrea ROLI.

Chapter 2

Boosting SLS using Resolution

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, modern systematic 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 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 structure such as symmetries, variable dependencies and clustering. This enhances the effect of deduction 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

Chapter written by ANBULAGAN, Duc Nghia PHAM, John SLANEY and Abdul SATTAR.

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.

In previous work [ANB 05] we enhanced a number of contemporary SLS solvers with a resolution-based preprocessor. Our experiments were limited to a small problem set and only one preprocessor. Although powerful and complex preprocessors are important to the success of many systematic SAT solvers, to date there has been no systematic evaluation of the effects of these preprocessors on SAT solvers, especially on local search techniques. In the study, we attempt to address this lack using problems, taken from SATLIB and SAT2005, known to be hard for SLS.

2.1. The SLS Solvers

Novelty [MCA 97], one of the best contemporary solvers in the WalkSAT family, may loop indefinitely due to its deterministic variable selection. To solve this problem, Hoos [HOO 99] added probabilistic random walks to Novelty, resulting in Novelty⁺. As with other WalkSAT variants, the performance of Novelty⁺ critically depends on the setting of its noise parameter, which controls the greediness of the search. AdaptNovelty⁺ addresses this problem by adaptively tuning its noise level based on the detection of stagnation. Later, Li and Huang [LI 05] 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 reducing randomness: WalkSAT first randomly picks an unsatisfied clause and then flips a variable of this clause. Their solver g2wsat uses a sophisticated gradient-based greedy heuristic to choose a promising variable from among those of all unsatisfied clauses.

Recently, clause weighting based SLS algorithms, most notably PAWS [THO 04] and SAPS [HUT 02], 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 the unsatisfied clauses, thus filling in the local minimum and forcing the search to move on. 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. These two mechanisms significantly increase the mobility of the search as well as helping to focus it on any “critical” clauses.

2.2. The Preprocessors

SAT Preprocessors are introduced to exploit hidden structure in problems. They 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 empirical study, we consider five state of the art preprocessors:

1) 3-Resolution. This restricted resolution procedure, used in the systematic solver Satz [LI 97], computes resolvents for all pairs of clauses of length ≤ 3 . Any resolvent of length ≤ 3 , is added to the formula. The procedure is repeated until saturation. Duplicates, subsumed clauses and tautologies are deleted.

2) 2-SIMPLIFY. This works on the implication graph of binary clauses in the formula, closing under transitive reduction, unit propagation and a restricted variant of hyper-resolution. The preprocessor was further enhanced by implementing subsumption and pure literal deduction rules [BRA 04].

3) HyPre. Like 2-SIMPLIFY, HyPre [BAC 04] reasons with binary clauses. It uses hyper-resolution to infer new binary clauses avoiding the space explosion of computing a full transitive closure. Unit and equality reductions are incrementally applied to infer more binary clauses, making the preprocessor more effective. It is more general and powerful than 2-SIMPLIFY.

4) NiVER. Variable Elimination Resolution (VER), used in the original Davis-Putnam (DP) procedure, replaces all clauses containing a chosen variable with their resolvents. Unfortunately, its space complexity is exponential. Non increasing VER (NiVER) [SUB 05] restricts the variable elimination to the case in which there is no increase in the number of literals after elimination.

5) SatELite. This extends NiVER with a Variable Elimination by Substitution rule. Several additions including subsumption detection and improved data structures further improve performance in both space and time [EÉN 05]. SatELiteGTI (a combination of SatELite preprocessor with MiniSAT solver) dominated the SAT2005 competition on the crafted and industrial problem categories.

2.3. Empirical Evaluation

We investigated the effect of preprocessing on SLS for 64 problem instances drawn from 12 classes of problems such as random 3-SAT, quasigroup existence and a wide range of industrial problems including job-shop scheduling, parity learning, and planning. 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 on a Linux Pentium IV computer with 3.0GHz CPU and 1GB RAM.

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. 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. 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.

2.3.1. Clause Weighting versus Random Walk

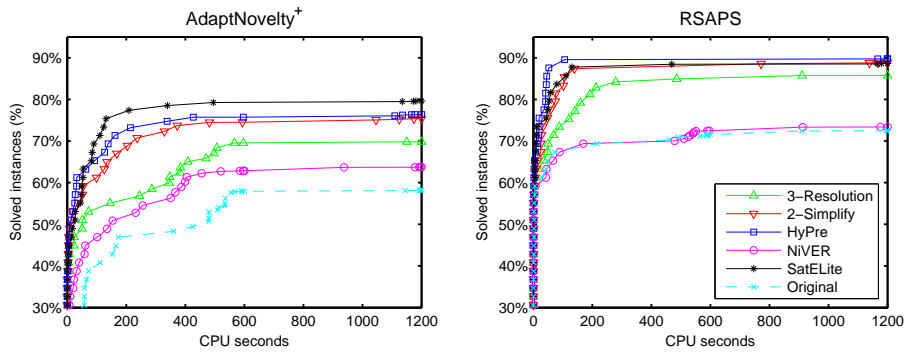


Figure 2.1. Runtime performance of resolution-enhanced SLS solvers.

Figure 2.1 shows the runtime distributions (RTDs) of *AdaptNovelty*⁺ and *RSAPS* over all industrial problems except *bmc* and *vpn*. The magnitude of improvement of solvers clearly depends on the heuristics employed in the solvers. For WalkSAT family solvers, e.g. *AdaptNovelty*⁺, *SatELite* is the promising choice and clearly outperforms the second option, *HyPre*. However, this pattern is reversed when applied to the clause weighting solver *RSAPS*. The improvement provided by *2-SIMPLIFY* to *RSAPS* is also very competitive to *SatELite*. Overall, the contribution of *3-Resolution* and *NiVER* to SLS are smaller and less reliable than those of *SatELite*, *HyPre* and *2-SIMPLIFY*. The poor performance of *NiVER* is expected, as it provides the least simplification on structured problem instances in our study.

2.3.2. Matching Preprocessors to Solver-Problem Pairs

In Table 2.1, where *PAWS*₁₀ is *PAWS* with parameter fixed to 10 and *RSAPS* is the reactive version of *SAPS*, we note the preprocessor that provides the most improvement on average for each 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. This is

Solvers\Problems	ais	bw	e*ddr	ferry	log	par16	qg	ssa	vmpc	3sat
AdaptNovelty ⁺	Sat	Sat	3-Res	Sat	HyP	Sat	HyP	Sat	HyP	NiV
g2wsat	Sat	HyP	Sat	Sat	HyP	Sat	HyP	Sat	Sat	Sat
PAWS ₁₀	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 2.1. *The best preprocessor for SLS-problem pair.*

perhaps not too surprising given that it is the most complex preprocessor. However, due to its novel implementation, SatELite records competitive runtimes in comparison with other preprocessors. HyPre is also valuable in most cases: it is the preferable choice of all four solvers to exploit the structures of the planning and quasigroup existence problems. It is worth noting that there is no uniform winner for the random 3-SAT problems. Indeed, each of the three solvers prefers a different preprocessor.

2.3.3. Multiple Preprocessing and Preprocessor Ordering

Instances	Preprocessor	#Vars/#Cls/#Lits	Ptime	Succ. rate	CPU Time		Flips	
					median	mean	median	mean
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 2.2. *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 2.2 shows preliminary results from an experiment with just one solver and two preprocessors on just two problems. We compare the effects of running each preprocessor separately, then of running SatELite after HyPre, and finally of running SatELite followed by HyPre.

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.

2.4. Conclusion

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.

2.5. Bibliography

- [ANB 05] ANBULAGAN, PHAM D. N., SLANEY J., SATTAR A., “Old Resolution Meets Modern SLS”, *Proceedings of 20th AAI*, p. 354–359, 2005.
- [BAC 04] BACCHUS F., WINTER J., “Effective Preprocessing with Hyper-Resolution and Equality Reduction”, *Revised Selected Papers of SAT 2003, LNCS 2919 Springer*, p. 341–355, 2004.
- [BRA 04] BRAFMAN R. I., “A Simplifier for Propositional Formulas with Many Binary Clauses”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, num. 1, p. 52–59, 2004.
- [EÉN 05] EÉN N., BIÈRE A., “Effective Preprocessing in SAT through Variable and Clause Elimination”, *Proceedings of 8th SAT, LNCS Springer*, 2005.
- [HOO 99] HOOS H. H., “On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT”, *Proceedings of 16th AAI*, p. 661–666, 1999.
- [HUT 02] HUTTER F., TOMPKINS D. A. D., HOOS H. H., “Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT”, *Proceedings of 8th CP*, p. 233–248, 2002.
- [LI 97] LI C. M., ANBULAGAN, “Look-Ahead Versus Look-Back for Satisfiability Problems”, *Proceedings of 3rd CP*, p. 341–355, 1997.
- [LI 05] LI C. M., HUANG W. Q., “Diversification and Determinism in Local Search for Satisfiability”, *Proceedings of 8th SAT, LNCS Springer*, 2005.
- [MCA 97] MCALLESTER D. A., SELMAN B., KAUTZ H. A., “Evidence for Invariants in Local Search”, *Proceedings of 14th AAI*, p. 321–326, 1997.
- [SUB 05] SUBBARAYAN S., PRADHAN D. K., “NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT Instances”, *Revised Selected Papers of SAT 2004, LNCS 3542 Springer*, p. 276–291, 2005.
- [THO 04] THORNTON J., PHAM D. N., BAIN S., FERREIRA JR. V., “Additive Versus Multiplicative Clause Weighting for SAT”, *Proceedings of 19th AAI*, p. 191–196, 2004.