Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Efficiently Exploiting Dependencies in Local Search for SAT

Duc Nghia Pham and **John Thornton** and **Abdul Sattar**
SAFE Program, NICTA Ltd., Australia and
Institute for Integrated and Intelligent Systems, Griffith University, Australia
{duc-nghia.pham, john.thornton, abdul.sattar}@nicta.com.au

Abstract

We propose a new local search platform that splits a CNF formula into three sub-components: i) a minimal dependency lattice (representing the core connections between logic gates), ii) a conjunction of equivalence clauses, and iii) the remaining clauses. We also adopt a new hierarchical cost function that focuses on solving the core components of the problem first. We then show experimentally that our platform not only significantly outperforms existing local search approaches but is also competitive with modern systematic solvers on highly structured problems.

Introduction

A basic question facing anyone interested in solving large and complex satisfiability (SAT) problems is whether to apply a local or systematic search solver. If we examine the relative performance of these solver types in recent SAT competitions (see http://www.satcompetition.org) the answer appears clear: local search (LS) is better for randomly generated problems and systematic search is better for structured real-world problems. Given that most practical problem encodings will reflect the regular structure of the real-world situations they represent, the ability of complete solvers to efficiently exploit structure argues strongly in their favour.

Recently, Pham, Thornton, & Sattar (2007) introduced a new *dependency lattice* (DL) platform that enables local search to effectively perform constraint propagation. The key idea involved extracting *dependent* variables from a CNF formula by searching for clausal structures that represent *logic gates* (Ostrowski *et al.* 2002). This extends the normal SAT solver framework to include a DL that models the extracted gates. Based on the new DL framework, a local search can be implemented that only flips the values of independent variables and dynamically calculates the effects of these flips on the overall solution cost. Using this approach, Pham, Thornton, & Sattar were able to construct the first local search solver able to reliably solve all five of the well-known par32 benchmarks within 24 hours.[1] While these were significant achievements, local search is still considerably slower than the best complete solvers in this area.

[1]Later, Prestwich (2007) also successfully solved the par32 problems using LS in conjunction with a new problem encoding.

A key reason for this inferior performance is that an all inclusive DL approach is not always suitable for handling different types of structures, as it can make the process of propagating constraints and calculating costs expensive. In contrast, the best systematic solver for the parity domain, march_eq (Heule & van Maaren 2004), does its equivalence checking in a conjunction of equivalences (CoE) formula separately from the main CNF formula. Another possible reason is that the DL approach has no effective way to deal with long chains of implication that often appear in industrial problems. It has been shown that local search performs poorly on problems containing this kind of implication chain (Wei & Selman 2002; Prestwich 2007).

In the rest of the paper, we propose a new *component* (COM) based approach for efficiently handling variable dependencies in local search that involves: i) decomposing an input problem into sub-components and ii) hierarchically solving these sub-components. We then show experimentally that our new platform significantly outperforms existing local search approaches and delivers performance that is now competitive with the best modern systematic solvers.

Decomposing into Sub-Components

Our approach for decomposing a CNF formula $\mathcal{F}$ into sub-components is performed in two steps. The first step involves simplifying $\mathcal{F}$ by iteratively performing propagation on all unary clauses and binary "eq/xor" clauses until saturation. We then decompose the simplified $\mathcal{F}$ into the following three initial sub-components: (i) a DL $\mathcal{L}$ that models all "and/or" gates extracted from $\mathcal{F}$; (ii) a CoE formula $\mathcal{E}$ that represents all "eq/xor" gates extracted from $\mathcal{F}$; and (iii) a new CNF formula $\mathcal{F}_r$ that contains all remaining clauses from $\mathcal{F}$ that do not represent any extracted gates. The new $\mathcal{E}$ formula is further simplified using the algorithm described in (Warners & van Maaren 2000) to avoid the issue of implication chains discussed above. Finding a satisfying assignment for $\mathcal{F}$ (after simplification) is now equivalent to finding an assignment that satisfies all "eq" clauses in $\mathcal{E}$ and all clauses in $\mathcal{F}_r$ whilst maintaining the consistency in $\mathcal{L}$.[2]

However, at this stage a straightforward application of local search to solve these sub-components will not have an advantage over the single DL-based approach. In fact, the

[2]Currently, $\mathcal{L}$ does not contain any external dependent gates.

situation can get worse as these sub-components are still tightly connected even after the decomposition (e.g. a dependent variable in $\mathcal{L}$ can be an independent variable in both $\mathcal{E}$ and $\mathcal{F}_r$). In such cases, a local search can be easily misdirected as it has no effective way to calculate the correct cost of flipping a variable.

Therefore, the second step of our approach attempts to move these kinds of dependencies from $\mathcal{E}$ and $\mathcal{F}_r$ back into $\mathcal{L}$ to enable constraint propagation. This is achieved as follows: for each clause $c = (x_1 \vee x_2 \vee \ldots \vee x_k)$ in $\mathcal{F}_r$, if c contains a dependent variable x_i either from $\mathcal{L}$ or $\mathcal{E}$ then we add an "or" gate $v = \vee(x_1, x_2, \ldots, x_k)$ to $\mathcal{L}$ and remove c from $\mathcal{F}_r$. In addition, for each dependent variable x_i in $\mathcal{E}$, if x_i is an input of a gate in $\mathcal{L}$ then we move the "eq" gate defining x_i from $\mathcal{E}$ to $\mathcal{L}$. It should be noted that finding a satisfying assignment for $\mathcal{F}$ (after simplification) is now equivalent to finding an assignment that satisfies all "eq" clauses in $\mathcal{E}$, all clauses in $\mathcal{F}_r$, and all external dependent gates in $\mathcal{L}$.

Solving with Sub-Components

The sub-components of a problem now present a local search solver with a set of variables, a cost calculation mechanism and the opportunity to treat each sub-component differently. Our approach starts with the recognition that the lattice $\mathcal{L}$ represents the most tightly connected sub-component and hence is likely to prove the most difficult to satisfy. In addition, performing constraint propagation in the dependency lattice is significantly more expensive than for the $\mathcal{E}$ or $\mathcal{F}_r$ formulas. Consequently, we decided to make it a priority to try and satisfy $\mathcal{L}$ by employing a *hard multiplier* strategy. This technique was previously used to bias a local search to try and satisfy hard (mandatory) constraints in over-constrained problems with hard and soft (preference) constraints (Thornton & Sattar 1998). A static hard multiplier associates a penalty multiplier m to a subset of clauses (known as multiplier clauses) at the start of a search. Then, when used in conjunction with a clause weighting algorithm, if a multiplier clause c_i attracts a weight of w_i during a search then the total weight on the clause becomes $m \times w_i$. In this way, weight is accumulated as if there were m copies of each multiplier clause. Additional techniques exist to dynamically alter the size of the multiplier to ensure that the total weight of any single multiplier clause will exceed the combined cost of all the currently false non-multiplier clauses.

In the current work, we associated a multiplier m with each external gate of $\mathcal{L}$ (these are the gates that determine the cost of flipping an independent variable in $\mathcal{L}$). While the option exists to implement a dynamic multiplier, we obtained consistently good results from setting m equal to half the number of non-DL clauses. We implemented the hard multiplier within two of the more successful SAT local search solvers: AdaptNovelty$^+$ (Hoos 2002) (as used in the original DL study) and gNovelty$^+$ (Pham *et al.* 2007) (the winner of the random SAT category in the SAT 2007 competition). Although gNovelty$^+$ has an additional clause weighting component, we turned this off for our experimental study, meaning the hard multiplier acted as a simple static bias towards satisfying the DL external gates.

Experimental Study

To evaluate the benefits of decomposition, we compared our new approach against the original DL-based approach reported in (Pham, Thornton, & Sattar 2007). To this end, we used the same ssa7552, parity 16- and 32-bit SATLIB benchmarks (available from http://www.satlib.org) reported in the original study. It should be noted that both problem classes decompose into only two sub-components (ssa into a lattice $\mathcal{L}_s$ and a CNF formula $\mathcal{F}_s$, and parity into a lattice $\mathcal{L}_p$ and a CoE formula $\mathcal{E}_p$).

Table 1 compares the performance of AdaptNovelty$^+$ and gNovelty$^+$ for both platforms.[3] As previously discussed, both solvers were augmented with a hard multiplier (when solving decomposed problems) that weights in favour of satisfying external DL gates. The Table 1 results report averages for 100 runs per instance on the ssa problems (with each run timed out after 1 hour) and 10 runs per instance on the par32 problems (with each run timed out after 24 hours). All experiments were performed on a machine with an AMD Opteron 252 2.6GHz processor with 2GB of RAM.

	AdaptNovelty$^+$		gNovelty$^+$	
Problem	#flips	#seconds	#flips	#seconds
ssa-038	$\frac{2,169}{2,065}$	$\frac{15.131}{0.106}$	$\frac{1,228}{1,337}$	$\frac{2.538}{0.072}$
ssa-158	$\frac{439}{430}$	$\frac{1.203}{0.002}$	$\frac{281}{285}$	$\frac{0.235}{0.001}$
ssa-159	$\frac{460}{436}$	$\frac{1.396}{0.005}$	$\frac{292}{234}$	$\frac{0.269}{0.003}$
ssa-160	$\frac{1,284}{1,283}$	$\frac{5.562}{0.047}$	$\frac{956}{1,062}$	$\frac{1.240}{0.039}$
par32-1	$\frac{n/a}{214,914}$	$\frac{48,194.033}{34.332}$	$\frac{109,742,764}{143,743}$	$\frac{25,378.797}{22.800}$
par32-2	$\frac{n/a}{2,301,746}$	$\frac{13,204.536}{360.955}$	$\frac{14,289,344}{476,059}$	$\frac{3,329.358}{73.962}$
par32-3	$\frac{n/a}{144,026}$	$\frac{17,766.822}{22.489}$	$\frac{57,127,508}{45,733}$	$\frac{13,079.765}{7.151}$
par32-4	$\frac{n/a}{4,626,512}$	$\frac{9,487.728}{726.010}$	$\frac{96,051,296}{3,189,435}$	$\frac{22,187.938}{494.928}$
par32-5	$\frac{n/a}{45,300}$	$\frac{23,212.755}{7.111}$	$\frac{113,712,655}{11,224}$	$\frac{26,507.983}{1.748}$

Table 1: The average performance of local search solvers on the *ssa* and *parity* problems shown in the form: $\frac{DL-based}{COM-based}$.

The results clearly show that the new COM-based decomposition technique produces dramatic performance benefits. Firstly, the ssa results illustrate the greater efficiency of the decomposition platform in terms of flip times, with each instance solved in roughly the same number of flips using the same algorithm, but with runtime speedups in excess of 100. In addition, gNovelty$^+$ is consistently better than AdaptNovelty$^+$ on the ssa instances, in terms of both flips and runtimes.

The parity results exceed the general level of improvement for ssa, with the additional effect that decomposed problems are solved in fewer flips, producing over $1,000$ times speedup on several instances. Bearing in mind that the best existing local search solvers can only solve the par32 problems within 12 to 24 hours, these results represent a significant breakthrough in the area.

However, our motivating question is whether a local search approach can compete with the best complete search

[3] Results on the parity 16-bit problems are omitted due to space limitations.

Solver	par32-1	par32-2	par32-3	par32-4	par32-5
AdaptNovelty$^+$	34.33	360.96	22.49	726.01	7.11
gNovelty$^+$	22.80	73.96	7.15	494.93	1.75
march_eq	0.19	0.26	1.29	1.34	3.44
lsat	52.96	26.23	78.95	36.97	180.86
eqsatz	90.37	22.48	1,001.57	86.79	1,396.14
dew_satz	8,603.08	3,528.65	4,225.39	6,983.15	62,636.26

Table 2: Runtimes (in seconds) of COM-based local search solvers and systematic solvers on the *parity* problems.

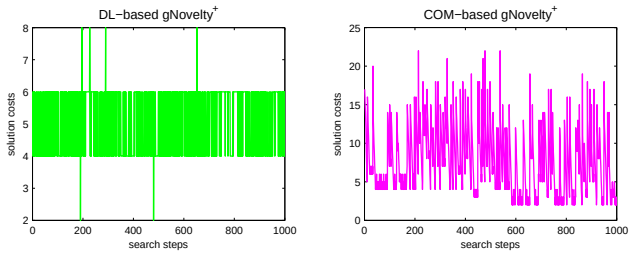

Figure 1: Search costs for the par32-5 instance.

techniques when solving highly structured problems. To this end, we solved the parity problems using four of the most effective complete solvers for this domain: march_eq (Heule & van Maaren 2004), lsat (Ostrowski *et al.* 2002), eqsatz (Li 2003) and dewsatz (Anbulagan & Slaney 2005). Table 2 compares these results with the COM-based runtimes for AdaptNovelty$^+$ and gNovelty$^+$. For the first time, these local search techniques are producing runtimes comparable with complete search. In particular, gNovelty$^+$ achieves the best average runtime of all the techniques considered on the par32-5 problem and on all par16 instances. It also achieves better runtimes than all other techniques except march_eq and lsat on the par32-2 and par32-4 instances.

Analysis and Conclusion

The results demonstrate, at least in the domain of the parity problems, that techniques to assist local search to recognize and exploit structure in CNF SAT instances can be competitive with the best complete search algorithms. The results also show that the exclusive use of a local search dependency lattice is not necessarily optimal and that the alternative of building a minimal lattice and handling other gates and clauses as separable components can yield dramatic performance benefits. While it is clear that the reduced size of the DL component in our decomposition leads to significant improvements in flip times (because the need for complex cost propagations in the lattice is eliminated or reduced) it is not so clear why decomposition should also reduce the number of flips needed to solve a problem. To investigate this, we graphed the local search costs of each technique on the par32-5 instance in Figure 1. These graphs show the number of false clauses at each flip and illustrate a lack of diversity in the behaviour of the DL-based runs, i.e. for most of the time the search is alternating between cost 4 and cost 6 solutions. Conversely, the COM-based plots show the searches visit a greater diversity of differing cost solutions. Clearly, the sub-component structure of the COM-based platform has created this greater diversification and (in this case) offers an explanation for the improved flip performance. We leave the question as to whether this is a one-off effect for the parity problems to further research.

In conclusion, we have extended the previous work on dependency lattices to improve the performance of local search on highly structured SAT CNF instances to a point where local search is now competitive with complete search on the well-known parity problem benchmarks. We have shown how the work on the conjunction of equivalences can be incorporated into a local search, and how the dependency lattice structure can be made more efficient via the decomposition of a CNF problem into three components. Finally, we have introduced a hard multiplier mechanism to bias the search towards satisfying external dependency lattice gates. In future work, there is scope to develop more sophisticated heuristics that can handle each problem component differently, and to implement a dynamic hard multiplier that can adjust itself to suit the search conditions.

Acknowledgments NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Anbulagan, and Slaney, J. K. 2005. Lookahead saturation with restriction for SAT. In *CP-05*, 727–731.

Heule, M., and van Maaren, H. 2004. Aligning CNF- and equivalence-reasoning. In *SAT-04*, 145–156.

Hoos, H. H. 2002. An adaptive noise mechanism for WalkSAT. In *AAAI-02*, 635–660.

Li, C. M. 2003. Equivalent literal propagation in the DLL procedure. *Discrete Applied Mathematics* 130(2):251–276.

Ostrowski, R.; Grégoire, É.; Mazure, B.; and Saïs, L. 2002. Recovering and exploiting strutural knowledge from CNF formulas. In *CP-02*, 185–199.

Pham, D. N.; Thornton, J.; Gretton, C.; and Sattar, A. 2007. Advances in local search for satisfiability. In *AI-07*, 213–222.

Pham, D. N.; Thornton, J.; and Sattar, A. 2007. Building structure into local search for SAT. In *IJCAI-07*, 2359–2364.

Prestwich, S. D. 2007. Variable dependency in local search: Prevention is better than cure. In *SAT-07*, 107–120.

Thornton, J., and Sattar, A. 1998. Dynamic constraint weighting for over-constrained problems. In *PRICAI-98*, 377–388.

Warners, J. P., and van Maaren, H. 2000. Recognition of tractable satisfiability problems through balanced polynomial representations. *Discrete Applied Mathematics* 99(1–3):229–244.

Wei, W., and Selman, B. 2002. Accelerating random walks. In *CP-02*, 216–232.