Comparing SAT preprocessing techniques

Lyndon Drake* lyndon@cs.york.ac.uk
Alan Frisch* frisch@cs.york.ac.uk
Inês Lynce† ines@sat.inesc.pt
João Marques-Silva† jpms@sat.inesc.pt
Toby Walsh* tw@cs.york.ac.uk

28 February 2002

1 Introduction

Preprocessing algorithms can dramatically improve the performance of even the most efficient SAT solvers on many problem instances [8]. Such algorithms simplify the formula by various means, including the deduction of necessary assignments, the addition of implied clauses, the deletion of redundant clauses, and the identification of equivalent literals. A number of preprocessors have been made publicly available, including `Compact` [2], `Compactor` [6], `Simplify` [9], and `CompressLite` [4]. While each preprocessor is very successful on some instances, they each combine a different selection of simplification techniques, making it difficult to understand which techniques are responsible for their success on a particular problem.

We are in the process of implementing each of the main simplification techniques in JQuest, a Java framework designed to support comparisons of SAT techniques. JQuest has already been used to evaluate the performance of various data structures for SAT solvers [7]. In addition to comparing existing techniques, we are creating efficient implementations of those techniques and investigating novel techniques. Here we briefly discuss two existing techniques: length-restricted resolution and binary equivalence finding.

2 Length-restricted resolution

One common technique, used for example in `Compactor` [6], is to do all possible binary resolutions where the resolvent has at most three literals. Other variations are possible, including limiting the size of the resolvents to two literals, and only doing resolutions where both parents will be subsumed. We also intend to investigate the effect on the branching heuristic of adding resolvents to the formula.

In practice, one problem with resolution is that resolvents are often subsumed by existing clauses in the formula. Adding redundant clauses is unlikely to make the instance simpler to solve, and so should be avoided if possible. We have a fast method for subsumption checking of binary and ternary clauses based on hash tables, which can be used to efficiently identify redundant resolvent clauses. We also have a similar technique, as yet unimplemented, for identifying and deleting clauses in the original formula if they are subsumed by a resolvent clause.

3 Binary equivalence finding

Some SAT instances, particularly those mapped from real world problems, contain equivalent literals [5]. For example, if a formula contains the clauses $(\neg a \vee b)$ and $(a \vee \neg b)$, we know that a and b

[1]University of York, Heslington, York YO10 5DD, United Kingdom.

This work and the first author are supported by EPSRC Grant GR/N16129 (see http://www.cs.york.ac.uk/aig/projects/implied).

[2]Technical University of Lisbon, IST/INESC/CEL, R. Alves Redol, 9, 1000-029 Lisbon, Portugal

are equivalent because a implies b and b implies a. Loops of binary equivalences are possible, such as $(\neg a \vee b)$, $(\neg b \vee c)$, and $(a \vee \neg c)$, in which a, b, and c are all equivalent. Given a set of equivalent literals, one of the literals in the set can replace all occurrences in the formula of the other literals in the set, without altering the satisfiability of the formula. This replacement reduces the number of variables in the formula (potentially reducing the number of assignments during the search), and can also identify tautological clauses (which can safely be removed from the formula).

The implications represented by binary clauses can be represented in a graph, in which sets of equivalent literals will appear as strongly connected components (SCCs). Tarjan [10] gives a linear time algorithm for finding SCCs in a graph, applied to SAT by Aspwall et al [1] and later de Val [3]. Because Tarjan's SCC algorithm is linear, binary equivalence finding is an particularly cost-effective technique.

4 Future work

A number of techniques still need to be implemented, in particular those that deduce necessary assignments. Once our implementation work has been completed, we intend to systematically compare the performance of the various preprocessing techniques. We also plan to theoretically study the relationships between the techniques.

5 Summary

Preprocessing techniques for SAT have not previously been experimentally compared in a systematic fashion. We are in the process of implementing many of the published techniques in a common platform (JQuest) in order to compare their performance.

References

[1] Aspwall, Plass, and Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, pages 121–123, March 1979.

[2] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 21–27, 1993.

[3] Alvaro de Val. Simplifying binary propositional theories into connected components twice as fast. In *LPAR'2001, 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science, pages 389–403. Springer-Verlag, 2001.

[4] Daniel le Berre. Exploiting the real power of unit propagation lookahead. In *LICS Workshop on Theory and Applications of Satisfiability Testing*, June 2001.

[5] Chu Min Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proceedings of AAAI-2000*, pages 291–296, Austin, Texas, USA, July 2000.

[6] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 1997.

[7] I. Lynce and J. Marques-Silva. Efficient data structures for fast sat solvers. Technical report, Instituto de Engenharia de Sistemas e Computadores, November 2001.

[8] Inês Lynce and João P. Marques-Silva. The interaction between simplification and search in propositional satisfiability. December 2001.

[9] João P. Marques-Silva. Algebraic simplification techniques for propositional satisfiability. In *International Conference on Principles and Practice of Constraint Programming*, pages 537–542, September 2000.

[10] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.