Adding resolution to the DPLL procedure for Boolean satisfiability

Lyndon Drake, Alan Frisch and Toby Walsh
{lyndon,frisch,tw}@cs.york.ac.uk
Department of Computer Science, University of York,
York YO10 5DD, United Kingdom

Abstract

We study the tradeoff between inference and search in the Davis Putnam algorithm. We show that neighbour resolution, a restricted form of resolution applied during search, can be simulated by applying binary resolution before search. We compare experimentally the cost of the two different methods. Our results demonstrate that binary resolution during preprocessing is generally effective at reducing both the size of the search tree and the total search time.

1 Introduction

The original Davis-Putnam [5] algorithm for propositional satisfiability (SAT) used resolution [12], but because of the space complexity of resolution [7] it was soon supplanted by DPLL [4], a backtracking search procedure. DPLL based search procedures have been the basis for almost all complete SAT solvers since then, including the fastest currently available SAT solvers [9, 10, 13].

Recently there has been interest in using resolution in combination with search, to reduce the amount of search required to solve SAT problems [11, 6]. The challenge involved in such techniques is ensuring that the cost of doing the resolution does not outweigh the benefit accruing to the search procedure. Resolution can prune the size of the search tree, but it is only worthwhile if the time spent on resolving is less than the time gained by reducing the search space.

We present two more techniques that combine resolution and search, and describe the relationship between them. The first technique, neighbour resolution, uses a restricted form of resolution during search, while the second uses a single level of binary resolution as a preprocessing step. We show that the preprocessing technique can prune the search tree as much as neighbour resolution during search, allowing us to replace expensive inference during search with a single stage of preprocessing.

2 Method A: Neighbour resolution during search

Neighbour resolution is a restricted form of resolution that produces resolvents which subsume both parents. We use neighbour resolution during complete search by applying it at decision points in the DPLL algorithm. By adding resolvents that subsume both parents, we can both prune the search tree and reduce the size of the formula. This is in contrast to most resolutions, which increase the size of the formula by adding more clauses.

Cha and Iwama [2] first used neighbour resolution to improve performance in a local search algorithm, ANC. By their definition, two disjunctive clauses are neighbouring iff they differ by the sign of a single literal. For example, $\{x \vee y \vee z\}$ and $\{\neg x \vee y \vee z\}$ are neighbours, while $\{a \vee b \vee c\}$ and $\{\neg a \vee d \vee e\}$ are not. Similarly to Cha and Iwama, we use the term "neighbour resolution" to refer to the resolution of neighbouring pairs of clauses.

Neighbour resolution was originally used between the restarts of the ANC local search algorithm. Neighbour resolutions are also valuable to a complete solver because they produce a resolvent that subsumes both of its parents. These resolvents can be used to simplify the formula by replacing a pair of clauses with one simpler clause. For example, resolving $\{x \vee y \vee z\}$ and $\{\neg x \vee y \vee z\}$ results in $\{y \vee z\}$, which subsumes both $\{x \vee y \vee z\}$ and $\{\neg x \vee y \vee z\}$.

The motivation for applying neighbour resolution during search is that, although SAT instances frequently contain very few neighbouring clauses, clauses often become neighbours as literals are deleted during search. For example, $\{x \vee y \vee z \vee a\}$ and $\{\neg x \vee y \vee z \vee \neg b\}$ are not neighbours, but would become neighbours if a was assigned false and b was assigned true.

3 Method B: Binary resolution before search

The second technique uses a single stage of binary resolution before starting the search procedure. All possible binary resolutions (except those that would produce a tautology) are carried out, and the resolvents added to the formula. Once added to the formula, resolvents are not used in further resolutions. In this section we show that a stage of binary resolution before search can have the same pruning effect on a search tree as neighbour resolution does during search.

3.1 Definitions

Two clauses are *resolvable* under binary resolution if they are of the form $\{w\} \cup X$ and $\{\neg w\} \cup Y$. We say that a pair of clauses are *neighbours* iff $X = Y$, and that they are *weak neighbours* iff $X \subseteq Y$. It is obvious that all neighbours are also weak neighbours, and that all weak neighbours are binary resolvable.

We refer to the ordered list of all assignments in effect at a particular node

n in a search tree by $\textsc{Assign}_n$. Assignments resulting from branching decisions, pure literal rule applications, and unit propagation are included in $\textsc{Assign}_n$, so $\textsc{Assign}_n$ is unique to node n. $\textsc{Assign}_n(c)$ means applying the assignments in their original order to a clause c.

Given a clause c, if in the course of a search we delete literals from c then we call the resulting clause c' a *descendant* clause, and we call c the *ancestor* of c'.

3.2 Proofs

Lemma 1. *Let c_1 and c_2 be two clauses at the root node of a search tree and assume that at some node n, $\textsc{Assign}_n(c_1)$ and $\textsc{Assign}_n(c_2)$ are neighbours and that their resolvent is c'. Then c_1 and c_2 are binary resolvable. Furthermore, if c is the resolvent of c_1 and c_2, then $\textsc{Assign}_n(c) = c'$.*

Proof. If $\textsc{Assign}_n(c_1)$ and $\textsc{Assign}_n(c_2)$ are neighbours, then from the definition of neighbourhood we know that $\textsc{Assign}_n(c_1) = \{w\} \cup X$ and $\textsc{Assign}_n(c_2) = \{\neg w\} \cup Y$, where $X = Y$. From the definition of $\textsc{Assign}_n$ we can see that $c_1 = \{w\} \cup X \cup A$ and $c_2 = \{\neg w\} \cup Y \cup B$, and A and B are the sets of literals deleted from c_1 and c_2 respectively by $\textsc{Assign}_n$. Therefore c_1 and c_2 are by definition binary resolvable.

The resolvent c' of $\textsc{Assign}_n(c_1)$ and $\textsc{Assign}_n(c_2)$ is $X \cup Y$. The resolvent c of c_1 and c_2 is $X \cup Y \cup A \cup B$. Applying $\textsc{Assign}_n$ to c will have the effect of deleting A and B, so $\textsc{Assign}_n(c) = X \cup Y$, and $\textsc{Assign}_n(c) = c'$. □

Theorem 1. *Consider two search trees, a and b, for a particular SAT instance. Both trees use the same branching order. Tree a is the result of applying neighbour resolution at each branching node. Tree b is the result of doing all binary resolutions before searching. In both trees, only descendants of clauses in the original formula (and not those that are derived from resolution) can be resolved. Let n_a be the number of nodes in tree a, and n_b be the number of nodes in tree b. Then $n_b \leq n_a$.*

Proof. As the branching order is the same in both trees, the only way that n_b could be greater than n_a is if a branch is pruned from tree a and not from tree b. Unit propagation does not affect the pruning, as any unit clause at node n in tree a will also be a unit clause by node n in tree b. Neither does the pure literal rule change the situation, because the same pure literals will appear in both trees. Both search trees apply to the same formula, so any such additional pruning in tree a could only be the result of a clause added to a by neighbour resolution which was not in b.

But by Lemma 1 we know that for any clause c_a added to a, another clause c_b has been added to b. If c_a was added at node n in tree a, then $\textsc{Assign}_n(c_b) = c_a$, and so any pruning resulting from c_a at node n will also occur in tree b. Because there is no way for a branch to be pruned from tree a without the same branch also being pruned from tree b, then $n_b \leq n_a$. □

3.3 Notes

We have demonstrated that any clause derived by neighbour resolution during search is the descendant of a clause that can be derived by binary resolution at the root of the search tree. In order to make this technique do everything that neighbour resolution does, we would have to subsume the parents of the resolvents whenever an assignment made it possible to do so. A second difference between the two techniques is that binary resolution not only generates the ancestors of the neighbour resolvents, but many other clauses as well.

4 Implementation

All our experiments were carried out using Swan, a SAT solver that we wrote in order to easily try out ways of using inference during search. Swan is written in C, and is reasonably fast (though it needs the addition of conflict clause recording to match the performance of the fastest available solvers).

In order to resolve clauses quickly, we applied an ordering of the literals to all the clauses (any ordering will do, so long as the literals are ordered the same way in all clauses). This makes it possible to produce a resolvent clause with a single pass over the clauses (i.e. in linear time in the length of the longest clause). Swan can do many thousands of resolutions per second on our test hardware. The time cost of each of the algorithms is almost entirely due to the time required to identify pairs of resolvable clauses (along with any subsequent time used in managing the added clause); the cost of actually resolving a pair of clauses is inconsequential.

In the initial preprocessing implementation, we identified resolvable clauses by comparing every pair of clauses in the problem. This was not cost effective, though it did prune the search tree well. An alternative method of finding such pairs is to use the positive and negative literal lists of each variable. This allows us to quickly find pairs of clauses with a common variable differing only by its sign. It is not always possible to resolve such clauses, as the resolvent might be a tautology, but we expected it to involve a smaller number of failed clause comparisons.

In practice, the real problem turned out to be the large number of added clauses. Many problems have so many resolvable clauses that doing all resolutions adds many more clauses than were in the problem initially. To get around this, we limited the number of resolutions so that the maximum number of clauses added was about the same number as in the original problem.

All the experiments were carried out on a dual Pentium III 750MHz with 256MB of RAM.

5 Results

5.1 Method A: Neighbour resolution during search

Using neighbour resolution at decision points during search was not an effective technique. While for many problems the search space was reduced dramatically (sometimes by two orders of magnitude), the cost involved in finding neighbours at many different points during the search was so high that it was rare for the solution time to decrease. We also looked at applying neighbour resolution at only some of the search nodes without success. The level of pruning on the search tree was encouraging, though, which is what led us to attempt to simulate neighbour resolution using binary resolution before search.[1]

5.2 Method B: Binary resolution before search

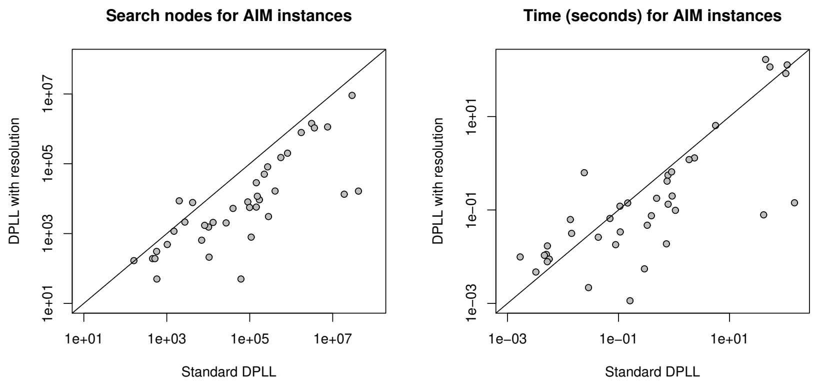

Figure 1: Performance of preprocessing binary resolution

This version was successful, both consistently pruning nodes and showing a time profit. Figure 1 shows a comparison between standard DPLL and DPLL with preprocessing binary resolution, from which it can be seen that DPLL with resolution nearly always wins. The problem instances in the graph are DIMACS AIM instances, and the times shown include the cost of the preprocessing. Points in the bottom right half of the graph are instances where DPLL with resolution performed better than standard DPLL. Note that, because the branching heuristic

[1]Our experiments with neighbour resolution allowed resolvents to become parents in subsequent neighbour resolutions. This does not correspond to the definition in Theorem 1, though we do believe that it had any significant impact on our results.

can see the added clauses, the number of search nodes is sometimes greater in the resolution version than in the standard DPLL version.

Interestingly, though, on one set of DIMACS problem instances (JNH), neighbour resolution lost in terms of both the number of search nodes and the search time. This suggests that different techniques may be necessary for different problem classes.[2]

6 Related work

Rish and Dechter [11] compared directional resolution (a form of ordered resolution) to search for SAT problems. Directional resolution makes it possible to identify pairs of resolvable clauses quickly. A heuristic for choosing a good ordering was described. They also showed that two hybrid algorithms combining resolution and search performed more efficiently than pure search (the first algorithm used directional resolution as a preprocessing step, while the second did resolution during search).

Van Gelder and Tsuji [6] describe `2cl`, a hybrid algorithm combining resolution and search. `2cl` replaces the unit propagation and pure literal operations of DPLL with a number of resolution and subsumption operations. They point out that efficient data structures are extremely important in practical hybrid algorithms. They show that their hybrid algorithm performs better than DPLL, and the performance improvement is greater for more difficult problems.

7 Future work

We intend to alter our preprocessing algorithm to more closely simulate the effect of neighbour resolution during search. One particularly valuable aspect of neighbour resolution is that the resolvents subsume both parents. Keeping track of all subsumptions resulting from assignments is too expensive, and so our current preprocessing algorithm ignores subsumption. We expect to be able to reduce the cost by only comparing resolvents with their parents during search, as these are the only subsumptions carried out by neighbour resolution. We also plan to limit the number of resolutions by using the branching order to predict which clauses are likely to become neighbours.

We will also compare the current algorithm with one where the heuristic is not allowed to consider inferred clauses. This is likely to be beneficial because inferred clauses cannot increase the number of search nodes unless the branching order is altered, which can only happen if the additional clauses are considered by the branching heuristic. Ensuring that the number of search nodes cannot

[2] More results will appear in the full paper.

increase as a result of adding clauses will limit the time cost to that of doing the resolutions.

One problem with using resolution as a preprocessing step is that some instances respond better than others (e.g. the differences we observed between the AIM and JNH problem classes). We intend to explore syntactic methods for detecting some of these instances, thereby avoiding attempting resolution on an instance where doing so is not worthwhile. A simple example of such a syntactic check is to consider problems where each clause consists of either all positive literals or all negative literals; it is impossible for a pair of such clauses to be neighbours. For example, $(a \vee b \vee c)$ and $(\neg a \vee \neg b \vee \neg c)$ cannot become neighbours as a result of assignments. It is simple to identify instances which possess this property, and avoid wasting time looking for resolvable pairs of clauses.

Perhaps the most interesting area we plan to look at is the interaction between conflict clause recording [8] and resolution-based simplification methods. Conflict clause recording is used by all of the best-performing SAT solvers available, and involves generating an implied clause at each backtrack point in order to prevent repeated exploration of dead regions of the search space. It may be possible to combine knowledge about the branching order and conflict recording algorithm to guide the choice of a preprocessing simplification algorithm.

A fifth possible line of research is to use neighbour resolution between restarts in a complete solver. Restarts have proved to be very effective in practice [1], and have also been theoretically shown to reduce expected solution times [3]. Neighbour resolution was originally applied between local search restarts with the intention of preventing the search procedure from reaching the same area of the search space again. Using neighbour resolution between restarts in a complete procedure might have the same effect.

8 Summary

We have presented two techniques for adding more inference to a complete search algorithm for SAT. The first of these techniques, neighbour resolution, is too expensive to be effective, despite the level of pruning it achieves. The second technique, a limited use of binary resolution as a preprocessing step, is cost effective. In addition, we have shown that binary resolution can be used to provide the same pruning effect as neighbour resolution. . We are planning a number of improvements to our implementation of the preprocessing algorithm, in particular to make it more closely follow the behaviour of neighbour resolution.

References

[1] L. Baptista, I. Lynce, and J. P. Marques-Silva. Complete search restart strategies for satisfiability. In *IJCAI Workshop on Stochastic Search Algorithms (IJCAI-SSA)*, August 2001.

[2] Byungki Cha and Kazuo Iwama. Adding new clauses for faster local search. In *Proceedings of AAAI-96*, pages 332–337, 1996.

[3] Hubie Chen, Carla Gomes, and Bart Selman. Formal models of heavy-tailed behaviour in combinatorial search. In Toby Walsh, editor, *Proceedings of CP 2001*, LNCS 2239, pages 408–421. Springer, 2001.

[4] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

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

[6] Allen Van Gelder. Satisfiability testing with more reasoning and less guessing. In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1995.

[7] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[8] João P. Marques Silva and Karem A. Sakallah. Conflict analysis in search algorithms for satisfiability. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, Nov 1996.

[9] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5), may 1999.

[10] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *39th Design Automation Conference*, Las Vegas, June 2001.

[11] Irina Rish and Rina Dechter. Resolution versus search: Two strategies for SAT. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000: Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 215–259. IOS Press, 2000.

[12] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.

[13] Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putman method. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000: Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 309–326. IOS Press, 2000.