Dew_Satz: Integration of Lookahead Saturation with Restrictions into Satz

Anbulagan
Logic and Computation Program, NICTA Ltd.
and
Computer Sciences laboratory, Australian National University
Canberra, Australia
anbulagan@nicta.com.au

1 Introduction

In this paper, we briefly describe the variants of Dew_Satz[1] [Anbulagan and Slaney, 2005], the complete backtrack search algorithm. It integrates lookahead saturation (LAS) with restriction techniques into the Satz[2] SAT solver [Li and Anbulagan, 1997a]. We use Satz215, the best version of Satz, in our experiments. The two restrictions used here are neighbourhood variable ordering (NVO) and dynamic equality weighting (DEW). Besides these two restrictions, we use also a static value ordering and a resolution-based preprocessor to create two (2) variants of Dew_Satz.

The variants are:

(1) Dew_Satz_1a: uses LAS, NVO, DEW, preprocessor, and static value ordering with ">" operation;

(2) dewSatz: uses LAS, preprocessor, and static value ordering with ">" operation.

2 Resolution-based Preprocessor

Resolution has been used as a component of propositional reasoning systems. It is sometimes used to enhance the performance of complete SAT solvers. The Satz solver [Li and Anbulagan, 1997b] used a restricted resolution procedure, adding resolvents of length ≤ 3, as a first phase process before running the complete backtrack search. This implementation gave modest performance gains (around 10%) on random 3-SAT problems, but was very important to the ability of Satz to solve many real-world benchmark problems.

When two clauses of a CNF formula have the property that some variable x_i occurs positively in one and negatively in the other, the resolvent of the clauses is a disjunction of all the literals occuring in the clauses except x_i and $\overline{x_i}$. For example, the clause $(x_2 \vee x_3 \vee \overline{x_4})$ is the resolvent for the clauses $(\overline{x_1} \vee x_2 \vee x_3)$ and $(x_1 \vee x_2 \vee \overline{x_4})$ and is added to the clause set. The new clauses, provided they are of length ≤ 3, can in turn be used to produce other resolvents. The process is repeated until saturation. Duplicate and subsumed clauses are deleted, as are tautologies and any duplicate literals in a clause. The resolution phase runs in polynomial time.

[1]available from http://rsise.anu.edu.au/∼anbu/
[2]available from www.laria.u-picardie.fr/∼cli/EnglishPage.html

2.1 Simple Decision Based Preprocessor

The resolution-based preprocessor can change the size of the initial formula by adding or eliminating clauses through resolution operations and subsumption. This includes the creation and discovery of unit clauses. Our observations of the preprocessing step showed that if very few unit clauses are discovered and the number of clauses in the formula grows, then the subsequent DPLL search is not effected or can even be hindered by the action of preprocessing.

3 The Heuristics

We present, in this section, four heuristics to enhance the branching heuristic used in Satz solver.

3.1 LAS

The basic idea of introducing an iterative lookahead process into DPLL is to detect failed literals earlier. Additionally, the lookahead process computes a variable's weight when there is no conflict found during two unit propagations. The variable with the highest weight will be selected as the next branching variable. Assume that, in the current iterative lookahead process, x_i has been selected as the branching variable candidate and the lookahead process on variable x_j finds a conflict from two unit propagations done. In this case, another successful unit propagation process on x_j will reduce the size of the formula. This action will affect the weight of any x_i such that x_i is a neighbour of x_j and $i < j$. For example, consider the clauses $(x_1 \vee \bar{x_3} \vee x_4) \wedge (\bar{x_1} \vee x_2) \wedge (x_1 \vee x_2) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4})$. The two unit propagations performed on variable x_1 result in $w(x_1)$=2 and $w(\bar{x_1})$=3. The iterative lookahead process next propagates on variable x_2, and a conflict is found during the propagation on $\bar{x_2}$. The conflict affects the weight of variable x_1, because variables x_1 and x_2 are neighbours. In the context of one iterative lookahead process, variable x_1 becomes a poor candidate for the next branching variable, because if it were chosen it could not prune the search tree efficiently.

Since the variable x_1's weight has changed, a lookahead saturation (LAS) based DPLL, studied in [Anbulagan, 2004], will enter the next iterative lookahead process. This process will continue until the weight of each branching variable candidate becomes stable. The key idea underlying LAS is to choose a branching variable which is really the best from an irreducible sub-formula at a given node of search tree. LAS is very similar to the "maintain arc consistency" (MAC) algorithm in CSP reasoning.

3.2 NVO

Since the LAS process is costly, the variables to be examined during the lookahead process, at the next node of the search tree, are restricted to the neighbourhood variables of the currently assigned variable. Note that, in the lookahead-based DPLL, a variable can be assigned during the lookahead process through unit clause reduction or when it is chosen as a branching variable. Variable x_i is a neighbour of x_j, if both variables occur in the same clause without considering their polarity, negative or positive. In Algorithm 1, procedure NVO(x_i) is executed to pile the neighbour variables of currently assigned variable into NVO_STACK. At the root of the search tree all variables in $\mathcal{V}$ will be piled into NVO_STACK.

Algorithm 1 DEW-NVO-LAS-BranchingRule($\mathcal{F}$)

```
 1: Push each variable $x_i \in \mathcal{V}$ to NVO_STACK at the root node;
 2: repeat
 3:   $\mathcal{B} := \emptyset$;  $\mathcal{F}_{init} := \mathcal{F}$;
 4:   for each free variable $x_i \in$ NVO_STACK do
 5:     Let $\mathcal{F}_i'$ and $\mathcal{F}_i''$ be two copies of $\mathcal{F}$;
 6:     if $w(x_i) = 0$ then
 7:       $\mathcal{F}_i' :=$ UP($\mathcal{F}_i' \cup \{x_i\}$);
 8:     end if
 9:     if $w(\bar{x_i}) = 0$ then
10:       $\mathcal{F}_i'' :=$ UP($\mathcal{F}_i'' \cup \{\bar{x_i}\}$);
11:     end if
12:     if empty clause $\in \mathcal{F}_i'$ and empty clause $\in \mathcal{F}_i''$ then
13:       return UNSATISFIABLE;
14:     else if empty clause $\in \mathcal{F}_i'$ then
15:       $\mathcal{F} := \mathcal{F}_i''$;  NVO($x_i$);
16:     else if empty clause $\in \mathcal{F}_i''$ then
17:       $\mathcal{F} := \mathcal{F}_i'$;  NVO($x_i$);
18:     else
19:       $\mathcal{B} := \mathcal{B} \cup \{x_i\}$;
20:       $w(x_i) := diff(\mathcal{F}_i', \mathcal{F})$;
21:       $w(\bar{x_i}) := diff(\mathcal{F}_i'', \mathcal{F})$;
22:       Compute_DEW($x_i$);
23:     end if
24:   end for
25: until $\mathcal{F} = \mathcal{F}_{init}$
26: for each variable $x_i \in \mathcal{B}$ do
27:   $\mathcal{W}(x_i) := w(x_i) * w(\bar{x_i}) + w(x_i) + w(\bar{x_i})$;
28: end for
29: NVO($x_i$);
30: return $x_i$ with highest $\mathcal{W}(x_i)$ to branch on;
```

3.3 DEW

We introduce dynamic equality weighting (DEW) to deal with the equality structure in some SAT problems. The basic concept of DEW is simple. Whenever the binary equality clause $x_i \Leftrightarrow x_j$, which is equivalent to 2 CNF clauses $\bar{x_i} \vee x_j$ and $x_i \vee \bar{x_j}$, occurs in the formula at a node, Satz needs to perform the lookahead process on x_i, $\bar{x_i}$, x_j, and $\bar{x_j}$. As result, variables x_i and x_j will be associated the same weight, (i.e. 3 following the computation at line 27 of Algorithm 1). Clearly, the processing of x_j and $\bar{x_j}$ is redundant, so avoid it by assigning the implied literal $\bar{x_j}$ (x_j's) the weight of its parent literal $\bar{x_i}$ (x_i's), and then by restricting the lookahead process to literals with weight zero. By doing so, we save two lookahead processes. The procedure Compute_DEW, in Algorithm 1, is called for weighting cumulatively the implied literals of the parent variable x_i. In our observations, this weighting scheme benefits the branching heuristic by choosing the a highest scoring branching variable. The literal's weight is also used to avoid redundant lookahead processes.

3.4 Static Value Ordering

The order in which a variable takes on a value at a branching point can effect the outcome for *satisfiable* problems. Usually, when the positive occurences of a variable is *greater than* the negative occurrences we instantiate the variable as *true* first, and vice-versa. This can be considered a *greedy* approach. We can also do the opposite. So when positive occurrence of a variable in current formula is *less than* its negative occurrence then we instantiate *true* first to the variable and vice-versa. This heuristic may be effective in anticipating satisfying assignments for which most of the *literals* in the given formula will be unsatisfied.

4 Contest Implementation

For the SAT 2007 Solver Competition, the Dew_Satz variants can accept up to 4,000,000 variables and up to 10,000,000 clauses.

Acknowledgments

We would like to thankfully acknowledge the financial support from the 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

[Anbulagan and Slaney, 2005] Anbulagan and John Slaney. Lookahead saturation with restriction for SAT. In *Proceedings of 11th CP*, pages 727–731, 2005.

[Anbulagan, 2004] Anbulagan. Extending unit propagation look-ahead of DPLL procedure. In *Proc. of 8th PRICAI*, pages 173–182, Auckland, New Zealand, August 2004. Springer, LNAI 3157.

[Li and Anbulagan, 1997a] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of 15th IJCAI*, pages 366–371, Nagoya, Aichi, Japan, 1997.

[Li and Anbulagan, 1997b] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proc. of 3rd CP*, pages 341–355, Austria, 1997.