TINISAT in SAT-Race 2006

Jinbo Huang
Logic and Computation Program
National ICT Australia
Canberra, ACT 0200 Australia
jinbo.huang@nicta.com.au

The development of TINISAT (pronounced "teeny sat") was started as part of an investigation of the effect of restart policies on clause learning (Huang 2006). The version entering the race, TINISAT 0.2, is written in about 760 lines of C++ including comments. The top-level procedure of the solver is given in Algorithm 1, which operates on an implicit CNF formula whose satisfiability is in question.

The following components of a typical clause learning SAT solver can be identified in Algorithm 1: decision heuristic (selectLiteral), unit propagation (decide, assertLearnedClause), clause learning (learnClause, backtrack), restarts (restartPoint, backtrack). We note that TINISAT 0.2 uses the 1-UIP (Zhang *et al.* 2001) learning scheme, does not delete clauses, and does not use randomness. The functions involved have the following semantics:

- selectLiteral uses some decision heuristic to select a free variable and then select one of its two literals, and returns it, or returns **nil** if no free variables exist.
- decide increments the decision level, sets the given literal to true, and performs unit propagation; it returns true iff no empty clause is derived.
- learnClause performs 1-UIP learning to derive an implicate of the CNF formula, and sets the assertion level (i) to 0 if the empty clause is derived, (ii) to 1 if a unit clause is derived, and otherwise (iii) to the second highest decision level among literals of the derived clause.
- assertionLevel returns the assertion level, which has been set by the last call to learnClause.
- restartPoint returns true iff the solver is to restart now according to some restart policy.
- backtrack(k) undoes all variable assignments in decision levels $> k$, and sets the decision level to k.
- assertLearnedClause adds the learned clause to the clause pool, performs unit propagation if the current decision level equals the assertion level (this is the condition under which the learned clause becomes unit), and returns true iff no empty clause is derived.

Algorithm 1 TINISAT

```
 1: loop
 2:   if (literal = selectLiteral()) == nil then
 3:     return SATISFIABLE
 4:   if !decide(literal) then
 5:     repeat
 6:       learnClause()
 7:       if assertionLevel() == 0 then
 8:         return UNSATISFIABLE
 9:       if restartPoint() then
10:         backtrack(1)
11:       else
12:         backtrack(assertionLevel())
13:     until assertLearnedClause()
```

Decision Heuristic TINISAT 0.2 uses the following decision heuristic: For each literal we keep a score that is initially the number of its occurrences in the original clauses. On learning a clause, we increment the score of every literal by 1 for each of its occurrences in clauses that are involved in the resolution process. The scores of all literals are halved once every 128 conflicts. When a decision is called for (Line 2 of Algorithm 1), we pick a (free) literal with the highest score from the most recently learned clause that has not been satisfied, and set it to true; if no such clause exists (at most 256 clauses are searched for this purpose) we pick any (free) literal with the highest score.

Restart Policy TINISAT 0.2 uses an instance of a class of restart policies proposed in (Luby, Sinclair, & Zuckerman 1993) based on the following sequence of run lengths: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots$, formally defined as the sequence $t_1, t_2, t_3, \ldots$ such that:

$$t_i = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1; \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \le i < 2^k - 1. \end{cases}$$

TINISAT 0.2 takes a "unit run" in this sequence to be 32 conflicts. Hence the actual restart intervals are: $32, 32, 64, 32, 32, 64, 128, \ldots$

References

Huang, J. 2006. The effect of restarts on the efficiency of clause learning. In *AAAI-06 Workshop on Learning for Search*.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47(4):173–180.

Zhang, L.; Madigan, C.; Moskewicz, M.; and Malik, S. 2001. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*.