TINISAT in SAT Competition 2008

Jinbo Huang
National ICT Australia and Australian National University
jinbo.huang@nicta.com.au

Overview

The development of TINISAT (Huang 2007a) started as part of an investigation of the effect of restart policies on clause learning (Huang 2007b). The version entering the competition, TINISAT 0.22, is coupled with the SATELITE preprocessor (Eén & Biere 2005). Algorithm 1 gives the top-level procedure of the solver.

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()
```

The following components of a typical clause learning SAT solver can be identified: decision heuristic (selectLiteral), unit propagation (decide, assertLearnedClause), clause learning (learnClause, backtrack), restarts (restartPoint, backtrack). Detailed semantics of the functions involved can be found in (Huang 2007b).

Decision Heuristic

TINISAT 0.22 uses the following decision heuristic: For each literal a score is kept that is initially the number of its occurrences in the original clauses. On learning a clause, the score of every literal is incremented 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 variable with the highest score (sum of two literal scores) from the most recently learned clause that has not been satisfied; if no such clause exists (at most 256 clauses are searched for this purpose) we pick any free variable with the highest score.

The variable is then set to a value using a heuristic inspired by (Pipatsrisawat & Darwiche 2006): Each variable has a field called *phase*, initially set to the value with the higher score. On backtracking, every variable whose assignment is erased has its *phase* set to that assignment, except for variables set in the latest decision level. When chosen for decision, a variable is set to its *phase*. However, this heuristic is bypassed if the two values differ in score by more than 32, in which case the value with the higher score is used.

Restart Policy

TINISAT 0.22 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:

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

TINISAT 0.22 takes a "unit run" in this sequence to be 512 conflicts. Hence the actual restart intervals are: $512, 512, 1024, 512, 512, 1024, 2048, \ldots$

References

Eén, N., and Biere, A. 2005. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 61–75.

Huang, J. 2007a. A case for simple SAT solvers. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, 839–846.

Huang, J. 2007b. The effect of restarts on the efficiency of clause learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2318–2323.

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

Pipatsrisawat, T., and Darwiche, A. 2006. SAT solver description: Rsat. In *SAT-Race*.