ManySat: solver description

Youssef Hamadi[1], Said Jabbour[2], and Lakhdar Sais[2]

[1] Microsoft Research
7 J J Thomson Avenue, Cambridge, United Kingdom
youssefh@microsoft.com
[2] CRIL-CNRS, Université d'Artois
Rue Jean Souvraz SP18, F-62307 Lens Cedex France
{jabbour,sais}@cril.fr

Overview

ManySat is a DPLL-engine which includes all the classical features like two-watched-literal, unit propagation, activity-based decision heuristics, lemma deletion strategies, and clause learning. In addition to the classical first-UIP scheme, it incorporates a new technique which extends the classical implication graph used during conflict-analysis to exploit the satisfied clauses of a formula [1].

When designing ManySat we decided to take advantage of the main weakness of modern DPLLs: their sensitivity to parameter tuning. For instance, changing parameters related to the restart strategy or to the variable selection heuristic can completely change the performance of a solver on a particular problem. In a multi-threading context, we can easily take advantage of this lack of robustness by designing a system which will run different incarnation of a core DPLL-engine on a particular problem. Each incarnation would exploit a particular parameter set and their combination should represent a set of orthogonal strategies.

The following components where used to differentiate each strategies:

- Variable selection
- Value selection, with a newly developed dynamic policy, and classical phase-learning [4, 5].
- Restarts, with newly developed dynamic policies.
- etc.

To allow ManySat to perform better than any of the selected strategy, conflict-clause sharing was added. This is done with respect to clause's size and to other factors. Technically, this is implemented in a way which minimizes locked accesses to a shared clause database.

Code

The system is written in C++ and has about 4000 lines of code. It was submitted to the race as a 32 bit binary. It is written on top of minisat 2.02 [3], which

was extended to accommodate the new learning scheme, the various strategies, and our multi-threading clause sharing policy. SatElite was also applied systematically by the treads as a pre-processor [2].

References

1. G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A generalized framework for conflict analysis. In *SAT (to appear)*, 2008.
2. Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.
3. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
4. Daniel Frost and Rina Dechter. In search of the best constraint satisfaction search. In *AAAI*, pages 301–306, 1994.
5. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In João Marques-Silva and Karem A. Sakallah, editors, *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.