SApperloT

Description of two solver versions submitted for the SAT-competition 2009

Stephan Kottler
Eberhard Karls Universität Tübingen, Germany
kottlers@informatik.uni-tuebingen.de

Abstract

In this paper we briefly describe the approaches realised in the two versions of SApperloT. The first version SApperloT-base primarily implements the state-of-the-art techniques of conflict-driven Sat-solvers with some extensions. SApperloT-hrp enhances the base version to a new hybrid three-phase approach that uses reference points for decision making.

1 The main issues of SApperloT

This chapter sketches the main ideas that are implemented in SApperloT-base and are also contained in SApperloT-hrp for the most parts. Both versions are complete Sat-solvers written in C++ using the functionality offered by the standard template library.

Solver basics SApperloT-base is a conflict-driven solver that implements state-of-the-art techniques like clause learning, non chronological backtracking and the two watched literal scheme that were originally introduced by GRASP [10] and CHAFF [11]. For the first version of SApperloT-base Minisat 2.0 [14, 4] was used as a guideline for efficient implementation.
Most decisions are made according to the previous assignment as in RSAT [12]. Moreover, we implemented the extension to the watched literal data-structure as described in [3]. Hence, instead of pointing directly from literals to clauses an indirection object is used. Binary and ternary clauses (both original and learnt clauses) are stored within this object. This has an impact (among other things) on the garbage collection of inactive learnt clauses since binary and ternary learnt clauses cannot be deleted using an activity value (which is applied for clauses with size > 3). To avoid the deletion of valuable long learnt clauses the garbage collection reduces the size of the learnts database by only one quarter and is therefore invoked more frequently. During the garbage collection the learnts database is split in two pieces by applying a variant of the linear median algorithm[1] not to waste time with sorting the learnts database at each call.

Activity values Many decisions in SApperloT are based on activity values like the VSIDS heuristic [11] and the garbage collection of learnt clauses. Also during the minimisation of learnt clauses [1, 14, 13] literals are ordered regarding their activity values. SApperloT-hrp uses the activity of clauses and variables even more extensive. To get the same results on different machines and with different optimization levels of the compiler we implemented a representation of activity values, as it is also done in PicoSAT [2].
Activity values are implemented as (restricted) fractions where the denominator is always a power of some predefined constant. Let $v = n/d$ be any activity value with $d = c^k$. The main operations done with activity related values are addition and multiplication. Since the results

[1] It is implemented in the function nth_element of the standard template library

of both operations will have a denominator $c^{k'}$ with some value k' the constant c can be omitted and just kept implicitly. For the above value v our data structure will just store the values n and k. This allows for storing very small numbers within a few bits. Since activity values are only used for relative comparison with each other small values are completely sufficient.
We use 16 bits for the nominator and 16 bits for the denominator for the activity values of variables and clauses. c is set to 128. Choosing, for instance, the reciprocal of the decaying factor as $r := 135/128$ and an initial activity addend as $a := 1/128^{65530}$ guarantees more than 6 million decay and add operations (s+=a; a*=r;) without having to perform an expensive decay of all activity values (worst case). Using the double size for activity values reduces expensive decay operations practically completely.

Preprocessing SApperloT does not perform any preprocessing on the input formula. Instead the solver has two features to simplify an instance during the solving process. At the first decision level always both polarities of a decision variable d are propagated. If there is a variable u that is assigned by unit propagation in both cases then either a unit clause is learnt (if u was assigned the same value twice) or two binary clauses can be learnt if they are not already contained in the formula [9, 8].
Moreover, after each period of 15 restarts asymmetric branching is applied for all clauses below average length. This helps to further shrink short clauses in order to prune the search space.

2 SApperloT-hrp – a hybrid version with reference points

Our motivation behind SApperloT-hrp is to develop a solver that utilises more information during the solving process and we intend to extend the solver by incorporating more structural information. The three-phase approach realised by the submitted version of SApperloT-hrp can be sketched as follows:

|base| Within this phase usual conflict-driven Sat-solving is applied. The solver gathers information about which clauses occur most frequently in conflicting assignments. Thus, we hold activity values for all clauses. If the solver cannot find a solution within a certain number of conflicts a subset $P \subseteq C$ of clauses is initialised holding the most active clauses.

|pcl| If P is a proper subset of C the solver aims to compute a model that satisfies all clauses in P. If a model is found the solver continues with the third phase. If P contains all clauses of C or if no model can be found within a certain number of conflicts the solver restarts (after simplification of the formula) with the first phase again. Obviously, if the clauses in P are unsatisfiable we conclude unsatisfiability of the entire formula.

|rp| If the solver enters this phase a model M is known that satisfies all clauses in P. This model is taken as a reference point for a variant of the DMRP approach [6, 7]. Thus, the solver tries to modify M so that all clauses in C are satisfied. If there are still some unsatisfied clauses $U \subset C$ after a certain number of conflicts a new set P is initialised: The new set P contains all clauses of U and the most active clauses of C. Also the size of P is remarkably increased and the solver continues with the second phase.

As already shown in [5] hybrid approaches can improve the performance of Sat-solvers. In SApperloT-hrp it seems that alternating the two phases |pcl| and |rp| gives the solver a quite good direction to find a solution for satisfiable instances or to resolve an empty clause if a formula is unsatisfiable. We first implemented an approximation of the break-count of variables as a basis for decisions in the DMRP approach. However, experiments showed that a fast and lazy implementation of the make-count of variables clearly outperforms the break-count approximation. We also achieved good speed-ups by optimising the data-structures to realise delta as defined in [7].
The current version of SApperloT-hrp already performs quite well on many families of instances. However, there are many parameters and magic constants that still have to be figured out by experiments.

References

[1] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *IJCAI*, pages 1194–1201, 2003.

[2] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 4:75–97, 2008.

[3] Geoffrey Chu, Aaron Harwood, and Peter J. Stuckey. Cache conscious data structures for boolean satisfiability solvers. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 6:99–120, 2009.

[4] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing - SAT 2003*, pages 502–518, 2003.

[5] Lei Fang and Michael S. Hsiao. Boosting sat solver performance via a new hybrid approach. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 5:243–261, 2008.

[6] Eugene Goldberg. Determinization of resolution by an algorithm operating on complete assignments. In *Theory and Applications of Satisfiability Testing - SAT 2006*, 2006.

[7] Eugene Goldberg. A decision-making procedure for resolution-based SAT-solvers. In *Theory and Applications of Satisfiability Testing - SAT 2008*, 2008.

[8] Marijn Heule. *SmArT Solving*. PhD thesis, Technische Universiteit Delft, 2008.

[9] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Principles and Practice of Constraint Programming*, pages 341–355, 1997.

[10] Joao P. Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.*, 48(5):506–521, 1999.

[11] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 530–535, 2001.

[12] Knot Pipatsrisawat and Adnan Darwiche. Rsat 2.0. In *Solver Description*, 2007.

[13] Niklas Sörensson and Armin Biere. Minimizing learned clauses. In *SAT 2009*, pages 237–243, 2009.

[14] Niklas Sörensson and Niklas Eén. Minisat - a SAT solver with conflict-clause minimization. In *Solver Description*, 2005.