SArTagnan & SApperloT

Description of the sequential solver SApperloT and the parallel solver SArTagnan

Stephan Kottler

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

SArTagnan

SArTagnan is a parallel portfolio SAT-solver that runs different algorithms and search strategies on different threads. The solver is implemented in C++ using OpenMP.

Clause sharing All threads are allowed to share clauses [BSK03, SLB05, HS09a] logically **and** physically. However, the set of clauses of different threads may differ and not all clauses have to be shared. One criterion to decide on which clauses to share is the LBD value [AS09]. However, in difference to the 2010 version of SArTagnan changes to the LBD value are only made locally within each thread. All sharing is generally realised without the use of mutex locks of the operating system.

Different strategies Most threads use CDCL [MS99] with the VSIDS heuristic [MMZ$^+$01] for variables [ES03], Luby restarts [LSZ93] or dynamic restarts as in Glucose [AS09] and phase-saving [PD07]. However, when run with 8 threads, three threads use geometric restarts and one thread uses activity values for literals as in the original VSIDS heuristic [MMZ$^+$01]. Most threads apply lazy hyper binary resolution as proposed in [Bie09].
Five out of 8 threads apply an extension to common Boolean Constraint Propagation as described in [KK11]. Clauses may be used for propagation even if they are not unit under the current partial assignment.

Sharing clauses physically allows for easily sharing different kinds of information among several threads. E.g. if one thread detects a clause for "on the fly improvement" [HS09b] all threads may benefit from this immediately. In this spirit two threads (when run with 8 threads) mainly attempt to improve the clause set for the other solvers:

One thread uses reference points for decision making (DMRP) as proposed in [Gol06, Gol08] and similar to [Kot10]. It frequently computes a reference point which attempts to reflect the search direction of several solvers: The value of any variable in a new reference point is set to the current predominant polarity when considering the assignments of all threads. Subsequently, the DMRP thread focuses on the set of clauses M that are not fulfilled by this reference point. M is examined in the order according to (shared) activity values of clauses. The activity is increased whenever a clause contributes to a conflict in any solving thread.

A particular thread frequently aims for simplifing the clause database. This thread applies common simplification techniques as variable elimination, clause subsumption and backward subsumption [EB05]. It also computes strongly connected components in the binary implication graph and detects and removes redundant binary clauses (shortcuts in the graph). Moreover, a variation of assymetric branching is applied frequently.

SApperloT

SApperloT is a sequential solver also written in C++. The submitted version of SApperloT mainly improves on the used data structure. It applies many of the simplification techniques of SArTagnan, however, it uses them less frequently. Moreover, it detects and removes blocked clauses [JBH10]. The new version of SArTagnan does not yet apply DMRP solving. This will be integrated again. However, it requires an adaption to the new data structure and was therefore not ready at the date of submission.
The version of SApperloT that uses the hybrid approach with DMRP solving [Kot10] and the preprocessor of Christian Zielke is submitted as SApperloT2010 (as it participated in the SAT Race 2010).

References

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *International Joint Conference on Aritifical Intelligence IJCAI*, pages 399–404, 2009.

[Bie09] Armin Biere. Lazy hyper binary resolution. Algorithms and Applications for Next Generation SAT Solvers, Dagstuhl Seminar 09461, Dagstuhl, Germany, 2009.

[BSK03] Wolfgang Blochinger, Carsten Sinz, and Wolfgang Küchlin. Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing*, 29(7):969–994, 2003.

[EB05] Niklas Eén and Armin Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *SAT*, pages 61–75, 2005.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, 2003.

[Gol06] Eugene Goldberg. Determinization of resolution by an algorithm operating on complete assignments. In *SAT 2006*, 2006.

[Gol08] Eugene Goldberg. A decision-making procedure for resolution-based SAT-solvers. In *SAT 2008*, 2008.

[HS09a] Youssef Hamadi and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT*, 2009.

[HS09b] HyoJung Han and Fabio Somenzi. On-the-fly clause improvement. In *SAT*, 2009.

[JBH10] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked Clause Elimination. In *TACAS*, pages 129–144, 2010.

[KK11] Michael Kaufmann and Stephan Kottler. Beyond Unit Propagation in SAT Solving. In *Symposium on Experimental Algorithms*, 2011.

[Kot10] Stephan Kottler. SAT Solving with Reference Points. In *SAT*, pages 143–157, 2010.

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. In *ISTCS*, pages 128–133, 1993.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *DAC*, 2001.

[MS99] J. Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *EPIA '99: Proceedings of the 9th Portuguese Conference on Artificial Intelligence*, pages 62–74, London, UK, 1999. Springer-Verlag.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.

[SLB05] Tobias Schubert, Matthew D. T. Lewis, and Bernd Becker. PaMira - A Parallel SAT Solver with Knowledge Sharing. In *MTV*, pages 29–36, 2005.