MiniLearningHeapExpSimp / SAT4J

Daniel Le Berre
CRIL-CNRS FRE2499, Université d'Artois, 62300 Lens, FRANCE

1 The SAT4J library

The SAT4J library [1] is an open source library of efficient SAT solvers in Java dedicated to people willing to embed SAT technology into their application without worrying about the details. SAT4J is currently used in model verification [10], ontology matching [5], requirements engineering [11], software product line configuration [2], etc. It started as a Java implementation of the MiniSAT specification [3], and evolved as a library when the solver was modularized to provide several heuristics, conflict analysis schemes, data structures, etc. One of the main feature in SAT4J solvers is the possibility to filter the constraints to learn after conflict analysis, since learning is not mandatory for backtracking, which is in contrast with usual conflict driven clause learning solvers [7].

The library can also handle cardinality or pseudo boolean constraints using full cutting planes reasoning. The library provides built-in CSP to SAT translators. Finally, a simple optimization scheme is provided, allowing basic MaxSAT and Weighted-MaxSAT solving.

2 The MiniLearning SAT solvers

MiniLearning inherits from Chaff [8] most of its features (first UIP learning, VSIDS heuristics, restarts strategy with increasing cutoff, etc.). It can selectively learn clauses, on syntactical basis (its length for instance) or on a more sophisticated heuristics (based on the activity of the variables). The version submitted to the SAT race, compared to the versions submitted to the SAT Competitions 2004 and 2005, inherits from last year winner MiniSAT 1.14 [4] both a heap based VSIDS heuristics and reason simplification during conflict analysis.

The main differences between MiniSAT 1.14 and MiniLearning (despite the programming language used) are the following:

backjumping The non chronological backtracking is done thanks to the first UIP conflict analysis scheme inherited from GRASP and Chaff but without having to learn a clause: backjumping and learning are decoupled in MiniLearning.

learning The asserting clauses found during conflict analysis are added to the current set of clauses or discarded according to their size or other heuristics. In the solver submitted to the SAT Race, only clauses of length less than 10% of the total number of variables are recorded. Note that if an asserting clause is not added to the set of clauses, it is recorded as the reason for propagating its unassigned literal, else the conflict analysis and backjumping scheme could not work. As a consequence, there is no gain in memory, rather a gain in the number of watched clauses.

heuristics As for VSIDS, each *variable* (not literal) has its activity increased if it appears in a learned clause, and divided by a constant periodically. That way, important variables should emerge after a few conflicts. In our settings, the activity is increased for each occurrence of a literal met during conflict analysis, while in MiniSAT it is done only once. The idea is to give more credit to literals that have been merged during resolution.

branching If a variable never appeared in a conflict, we branch first on its negative literal, else we branch on the literal that appeared in the last learnt clause. This is an easy way to implement that latest learnt clauses should be satisfied, instead of keeping a record of those clauses as in Berkmin[6].

reason simplification Once conflict analysis is performed using the First UIP scheme, the derived clause is simplified using the recursive approach proposed in MiniSAT 1.14[4], without the abstraction on decision levels. That simplification has a side effect in MiniLearning since reducing the size of the clauses will increase the chances for a given clause to be learnt. Note that only a light version of that simplification was available in the versions used for the qualification (MiniLearningHeapEZSimp).

memory/clause management Learnt selected clauses are kept as much as possible. The solver removes half of the clauses learnt (the less active) when available memory is under 5 MB. This is in contrast with most SAT solvers that clear learnt clauses periodically. We got mixed results using that strategy on medium size benchmarks, but it is the best option for us to deal with huge benchmarks.

Note that the library contains more than 20 pre-built SAT solvers, with various options enabled. Not all of them are compatible: it is for instance impossible for the moment to enable reason simplification with the specific structure to handle binary clauses proposed in [9]. The SAT solver submitted to the competition is the one that provided the best results on the tests sets.

3 Expected behavior

We expect SAT4J to show reasonable performances during the SAT Race. We did not introduce any specific heuristics for solving the test sets, so the solver should have a similar behavior during the race. The solver should perform a bit better during the race than during the qualifications since the full reason simplification was not implemented in the solvers submitted for the qualifications. That feature gave better results on the race test sets. The new upcoming Java Virtual Machine (Java 6 "Mustang" beta 2) provides a 10% to 20% speedup compared to the JVM used for the SAT Race (Java 1.5.0_04).

References

[1] SAT4J: The SATisfiability library for Java, 2004. http://www.sat4j.org/.

[2] Don Batory. Feature models, grammars, and propositional formulas. In *Proceedings of SPLC 2005*, volume 3714 of *LNCS*, pages 7–20, 2005.

[3] Niklas Eén and Niklas Sörensson. An extensible sat solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing*, LNCS 2919, pages 502–518, 2003.

[4] Niklas Eén and Niklas Sörensson. Minisat 1.14, a sat solver with conflict-clause minimization. SAT Competition 2005 Solver Description, 2005.

[5] Fausto Giunchiglia, Mikalai Yatskevich, and Enrico Giunchiglia. Efficient semantic matching. In *Proceedings of ESWC'05*, volume 3714 of *LNCS*, pages 272–289, 2005.

[6] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, March 2002.

[7] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[8] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, June 2001.

[9] Lawrence Ryan. Efficient algorithms for clause learning sat solvers. Master's thesis, SFU, February 2004. Available at http://www.cs.sfu.ca/ mitchell/papers/ryan-thesis.ps.

[10] Emina Torlak. Kodkod: a sat-based model finder for first order logic with relations, transitive closure, and partial instances, 2006. http://www.mit.edu/people/emina/kodkod.html.

[11] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, and John Mylopoulos. Requirements-driven configuration of software systems. In *Proceedings of RETR'05*, November 2005. http://www.cs.toronto.edu/km/retr/camera/yu05retr.pdf.