SAT4JPseudo: replacing resolution by cutting planes

Daniel Le Berre and Anne Parrain

Abstract

SAT4J [1] is an open-source library of conflict-driven clause learning SAT solvers in the spirit of GRASP, zChaff and MiniSAT in Java. Its extension to pseudo-boolean optimization is done by replacing the resolution performed between clauses by cutting planes, in the spirit of PBChaff [5] or Galena [4]. Compared to the version submitted to the PB05 evaluation, the solver is representing each kind of constraints specifically, instead of translating everything into pseudo boolean constraints. Thus, cutting planes can be performed on any combination of them. Furthermore, some simplifications inherited from PBChaff have been added to reduce the cost of applying cutting planes for conflict analysis. Each kind of constraints is represented specifically: clauses and cardinality constraints using watched literals, pseudo boolean constraints using counters. Finally, one version of the solver uses the objective function to guide the heuristics in optimization problems.

From clauses to constraints

The main issue when dealing with Pseudo Boolean constraints and cutting planes in a conflict driven solver such as GRASP or Chaff is the conflict analysis and learning scheme. Indeed, those solvers need to learn a new constraint each time a conflict is met. Furthermore, the new constraint must propagate some variable truth values. While those solvers are certainly the best framework for solving pure SAT problems, it is not clear that a similar approach is really appropriate for other kind of constraints. After the first pseudo boolean evaluation, it was not clear if extending conflict driven clause learning solvers for pseudo boolean constraint was a good approach. The aim of submitting the SAT4JPseudo solver in the second pseudo boolean evaluation is to provide a reasonably efficient and tuned full-cutting-plane-based solver extending existing CDCL solvers. Unlike solvers such as PBS [2] and Pueblo [6], also based on a CDCL framework, the idea is to keep the full power of cutting planes, without any tradeoff for efficiency (clause or hybrid learning, fixed-size integer arithmetic). There might be smarter ways to implement such approach, so poor performances of SAT4JPseudo should not mean that CDCL extended with cutting plane are not good for PBO.

Basically, the algorithm implemented in SAT4JPseudo to ensure to obtain conflictual constraint after cutting plane, as well as the algorithm to detect assertive constraints are the ones described in the article by D. Chai and A. Kuehlmann [4]. The only difference is that SAT4JPseudo uses the following property presented by H. Dixon [5] : Let c_1 and c_2 be two pseudo-boolean constraints such that c_1 propagates the literal y and c_2 propates $\neg y$, and the coefficient of y is 1. Then the result of applying a cutting plane between c_1 and c_2 is a falsified constraint. This property avoids some computations, especially when clauses or cardinalities are involved in cutting planes.

Dealing with specificities

Clauses and cardinality constraints can be seen as a special case of linear pseudo boolean constraints. Pseudo-boolean constraint $l_1 + l_2 + \ldots + l_n \geq 1$ translates to the clause $l_1 \vee l_2 \vee \ldots l_n$; $l_1 + l_2 + \ldots + l_n \geq k$ translates to the cardinality constraint $atleast(k, \{l_1, l_2, \ldots, l_n\})$. Pseudo-boolean constraints offer more expressivity than boolean constraints, and cutting plane inference is stronger than

resolution. But the cost for maintaining the data structures needed to represent such constraints is important (see e.g. [5, 4] for a discussion regarding the watched literals scheme for each kind of constraints). This is even worst when dealing with arbitrary precision integers (so-called *big integers*). Indeed, since cutting plane operations involve multiplying coefficients, one can easily reach an overflow with fixed length integers.

One way to reduce this cost is to really implement each kind of constraint (clause, cardinality, or pseudo-boolean constraint) in a specific way. This approach has two advantages : (i) unit propagation is more efficient since performed with minimal overhead; (ii) there is a limited use of big integers for clauses or cardinalities. It main drawback is to prevent shortcuts in the code to make the solver more efficient.

The way a constraint are represented is decided either when reading the instance after the normalization of the constraint, in the sense of P. Barth [3], or when a new constraint is derived at the end of the conflict analysis process.

The PB05 version of SAT4JPseudo was representing all the constraints as pseudo boolean constraints. If watched literals based clauses and cardinalities provide better results than counter based, it seems to be the opposite for counter based and watched literals based pseudo-boolean constraints [4]. As a consequence, we use counter based pseudo-boolean constraints.

Optimization problems

The optimization function is handled by adding a new pseudo boolean constraint in the solver each time a solution is found (with a degree equals to the solution's value minus one). In the first pseudo boolean evaluation, SAT4JPseudo was among the solvers with the best upper bounds, which is surprising considering the simple way optimization is handled. For this second evaluation, we also submitted a solver with its heuristics initialized in order to minimize the cost function: the variables of the cost function see their initial truth value set to true if its coefficient is negative, else it is set to false.

Conclusion

The new release of SAT4JPseudo has been fine tuned to behave as good as possible for a pure cutting-plane-based solver. SAT4JPseudo is in theory able to solve problems that are out of scope of pure resolution-based solvers such as PBS or MiniSAT+, or mixed resolution/cutting-planes such as Pueblo (Pigeon-hole like problems for instance). However, the cost of using arbitrary precision integers is not negligible (almost one third of the CPU-time). Detecting efficiently assertive constraints is still an issue. SAT4JPseudo is expected to perform reasonably on mixed CNF/cardinality/PB problems, and to perform better on pure PB problems.

References

[1] SAT4J, a SATisfiability library for java. http://www.sat4j.org.

[2] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Generic ILP versus Specialized 0-1 ILP: an update. In *Proceedings of ICCAD'02*, pages 450–457, 2002.

[3] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI–I–95–2–003, Max-Plank-Institut fr Informatik, Saarbrücken, 1995.

[4] Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *ACM/IEEE Design Automation Conference (DAC'03)*, pages 830–835, Anaheim, CA, 2003.

[5] Heidi Dixon. *Automated Pseudo-Boolean Inference within the DPLL framework*. PhD thesis, University of Oregon, 2004.

[6] Hossein M. Sheini and Karem A. Sakallah. Pueblo: A Hybrid Pseudo-Boolean SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:165–182, 2006.