SAT Solver Description: Rsat

Thammanit Pipatsrisawat and Adnan Darwiche

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
{thammakn,darwiche}@cs.ucla.edu

Abstact

Rsat is a DPLL–based solver that inherits many of the effective techniques introduced by MiniSat [3, 4] and Zchaff [6]. Rsat employs three improvements that have proven quite effective empirically. First, it employs techniques for reducing the time spent on Boolean constraint propagation. Next, it introduces a new technique that enables the solver to better focus on subproblems. Finally, Rsat employs an improved phase selection technique that has been designed to prevent the solver from repeating work it has already done upon backtracking.

1. Basic Algorithms and Data Structures

Rsat was originally developed as a SAT solver to be embedded in exhaustive search algorithms used by model counters and knowledge compilers [2]. In particular, the solver was engineered to enable a recursive high–level structure, with primitives that provide flexible control over its search procedure (non–chronological backtracking in particular). Rsat includes an iterative interface too, which is more appropriate for standard SAT solving as it avoids overflowing the runtime stack when solving very large problems.

Rsat uses a standard conflict–directed backtracking mechanism, as employed by Zchaff, with a clause learning scheme that is very similar to the one employed by MiniSat [4]. In particular, learned clauses are derived from implication graphs using the FirstUIP method [7], and then strengthened further using the conflict minimization technique described in [1]. The decision heuristic in Rsat is also similar to the one used by MiniSat, but its branching heuristic is different and is discussed later. The data structures used in Rsat are based on the two–watched literal scheme [6]. Rsat puts a special emphasis on using lightweight data structures to minimize memory usage. It tries to enhance cache performance by avoiding special structures for variables and literals. Instead, similar attributes are grouped together as they are often accessed at the same time.

2. Prioritized Implication Queue

Rsat manages the implication queue in a slightly different manner from conventional methods. Instead of dequeuing the first literal in the queue that was implied, Rsat uses the same heuristic as the decision heuristic to select a variable from the implication queue to process next. This idea was largely inspired by the work described in [5]. We experimented with several variations of this algorithm and the above algorithm seemed to yield the best performance.

The main reason behind prioritization is to cut down the amount of time used to perform Boolean constraint propagation. The hope is that, with prioritization, more constrained variables will tend to be processed first and, hence, conflicts, if they exist, will be detected faster. Prioritizing the implication queue should also help the solver to focus on a particular part of the problem better, because conflicting clauses encountered will tend to contain literals that were involved in recent conflicts. Intuitively, this should have positive effects on the performance of the solver, because staying focused on a subproblem will allow the solver to make the most use of learned clauses and help it derive strong clauses faster.

3. Improved Phase Selection Heuristic

Rsat uses an unconventional phase selection heuristic that was designed to avoid solving the same part of the problem multiple times. Our empirical results show that SAT solvers based on the standard DPLL framework could end up solving certain subproblems multiple times due to backtracking. Even if some of these subproblems can be solved easily the first time, there is no guarantee that they will continue to be easily solved in a different state of the solver.

To overcome this problem, every time Rsat backtracks, it remembers every assignment it erases. Rsat then uses these saved assignments in making further decisions. This technique significantly improves the performance of Rsat, as it helps in avoiding situations where the solver keeps

erasing the progress it has made thus far.

4. References

[1] BEAME, P., KAUTZ, H., AND SABHARWAL, A. Understanding the power of clause learning. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-2003)* (2003).

[2] DARWICHE, A. New advances in compiling CNF to decomposable negation normal form. In *Proceedings of European Conference on Artificial Intelligence* (2004), pp. 328–332.

[3] EÉN, N., AND SÖRENSSON, N. Minisat a sat solver with conflict-clause minimization. *The International Conference on Theory and Applications of Satisfiability Testing* (2003).

[4] EÉN, N., AND SÖRENSSON, N. Minisat a sat solver with conflict-clause minimization. *The International Conference on Theory and Applications of Satisfiability Testing* (2005).

[5] LEWIS, M. D. T., SCHUBERT, T., AND BECKER, B. W. Early conflict detection based bcp for sat solving. *The International Conference on Theory and Applications of Satisfiability Testing* (2004).

[6] MOSKEWICZ, M., MADIGAN, C., ZHAO, Y., ZHANG, L., AND MALIK, S. Chaff: Engineering an efficient sat solver. *39th Design Automation Conference (DAC)* (2001).

[7] ZHANG, L., MADIGAN, C., MOSKEWICZ, M., AND MALIK, S. Efficient conflict driven learning in a boolean satisfiability solver. *Proceedings of ICCAD 2001* (2001).