zChaff SAT Solver

Zhaohui Fu   Yogesh Marhajan   Sharad Malik
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544, USA
{zfu,yogism,sharad}@Princeton.EDU

I. Introduction

zChaff is a SAT solver that targets the industrial category and hopes to be reasonably successful in the handmade category. It implements the well known Chaff algorithm [1] which includes the innovative VSIDS decision strategy and a very efficient Boolean constraint propagation procedure. zChaff is a popular solver whose source is available to the public. It is possible to compile zChaff into a linkable library for easy integration with other applications and successful integration examples include the BlackBox AI planner [2], NuSMV model checker [3], GrAnDe theorem prover [4], and others. Performance-wise, zChaff compares well with other SAT solvers – versions of zChaff have emerged as the Best Complete Solver in the industrial and handmade instances categories in the SAT 2002 Competition [5] and as the Best Complete Solver in the industrial category in the 2004 SAT Competition [6].

II. Overview of the zChaff Solver

We will present a quick overview of the main features of the zChaff solver. The detailed information can be found at [7].

A. Decision Strategy - VSIDS

The Chaff [1] solver proposed the use of a heuristic called Variable State Independent Decaying Sum (VSIDS). VSIDS keeps a score for each literal of a variable.

B. Boolean Constraint Propagation - Two Literal Watching

zChaff uses the Two Literal Watching scheme [1] for BCP as proposed by Chaff [1]. A key benefit of the two literal watching scheme is that at the time of backtracking, there is no need to modify the watched literals in the clause database. This reduces the total number of memory accesses.

C. Conflict Driven Clause Learning and Non-chronological Backtracking - Learning the FirstUIP Conflict Clause

Conflict driven clause learning along with non-chronological backtracking were first incorporated into a SAT solver in GRASP [8] and relsat [9]. These techniques are essential for efficient solving of structured problems.

D. Increased Search Locality

When VSIDS was first proposed, it turned out to be very successful in increasing the locality of the search by focusing on the recent conflicts. However, recent experiments show that branching within greater locality helps dramatically to prune the search space.

1) Variable Ordering Scheme for VSIDS: This is the default decision heuristic for zChaff. One way of trying to make VSIDS more local is to increase the frequency of score decay.

2) BerkMin Type Decision Heuristic: The use of the most recent unsatisfied conflict clauses as is done by BerkMin also turns out to be a good cost-effective approach to estimate the locality. The main ideas of this approach are described by the authors of BerkMin [10].

3) Conflict Clause Based Assignment Stack Shrinking: This is related to one of the techniques used by the Jerusat solver [11]. When the newly learned firstUIP clause exceeds a certain length L, we use it to drive the decision strategy.

E. Learning Shorter Clauses

Short clauses potentially prune large spaces from the search. They lead to faster BCP and quicker conflict detection. Conflict driven learning derives new (conflict) clauses by successively resolving the clauses involved in the current conflict.

1) Short Antecedent Clauses are Preferred: When the clauses do not share many common literals, the sum of the lengths of all the involved clauses will determine the length of the learned conflict clause. We can directly influence the choice of clauses for the resolution by preferring shorter antecedent clauses.

2) Multiple conflict analysis: This is a more costly technique than replacing antecedents. It is often observed that BCP returns not one but many conflicting clauses (most of which are derived from some common resolvents). For each conflicting clause, zChaff finds the length of the firstUIP clause to be learned, and only records the one with the shortest length.

F. Aggressive Clause Deletion

As in BerkMin, zChaff periodically deletes some learned clauses using usage statistics and clause lengths to estimate the usefulness of a clause.

G. Frequent Restarts

Luck plays an important role in determining the solving time of a SAT solver. zChaff also uses a rapid fixed interval restart policy. The frequent restarts are observed to make the solver more robust.

References

[1] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an Efficient SAT Solver,” in *38th DAC*, 2001.
[2] http://www.cs.washington.edu/homes/kautz/blackbox/.
[3] NuSMW Home Page, http://nusmv.irst.itc.it/.
[4] GrAnDe, http://www.cs.miami.edu/~tptp/ATPSystems/GrAnDe/.
[5] SAT Competition 2002, http://www.satlive.org/SATCompetition/2002/.
[6] SAT Competition 2004, http://www.satlive.org/SATCompetition/2004/.
[7] Y. S. Mahajan, Z. Fu, and S. Malik, “Zchaff2004: An efficient sat solver.” *Lecture Notes in Computer Science: Theory and Applications of Satisfiability Testing, 7th International Conference, Invited Paper*, vol. 3542, pp. 360–375, 2005.
[8] J. P. Marques-Silva and K. A. Sakallah, “GRASP - A New Search Algorithm for Satisfiability,” in *IEEE International Conf. on Tools with Artificial Intelligence*, 1996.
[9] R. Bayardo and R. Schrag, “Using CSP look-back techniques to solve real-world SAT instances,” in *National Conference on Artificial Intelligence (AAAI)*, 1997.
[10] E. Goldberg and Y. Novikov, “BerkMin: A Fast and Robust SAT Solver,” in *DATE*, 2002.
[11] A. Nadel, “The Jerusat SAT Solver,” Master’s thesis, Hebrew University of Jerusalem, 2002.