New Features of the SAT'04 versions of zChaff

Zhaohui Fu Yogesh Mahajan Sharad Malik
Department of Electrical Engineering
Princeton University
{zfu,yogism,sharad}@EE.Princeton.EDU

1 Introduction

Zchaff is an implementation of the well known Chaff algorithm [6]. It participated in the SAT 2002 Competition and won the Best Complete Solver in both industrial and handmade benchmark categories. It is a popular solver and can be compiled into a linkable library for easy integration with user applications. Successful integration examples include the BlackBox AI planner [1], NuSMV model checker [2], GrAnDe theorem prover [3], and others.

2 New Features

This year's competition entries continue to use many features from the original version like the Variable State Independent Decaying Sum (VSIDS) scores for decision making, the Two Literal Watching scheme for boolean constraint propagation and non-chronological backtracking with firstUIP conflict analysis. These features are well-known and are widely adopted by many other DPLL based state-of-the-art SAT solvers. The main features of the new versions are adoption of a rapid restart policy, a more locality centric decision strategy, multiple conflict analysis, and a more aggressive clause database management.

2.1 Increased Search Locality

When it was proposed, VSIDS turned out to be very successful in increasing the locality of the search and this was observed to lead to faster solving times. Though VSIDS scores are biased towards recent regions of the search by the decaying of the scores, the decisions made are still *global* decisions, which is consistent with the fact that the variable score only stores the global information. However, recent experiments show that branching on the variables within certain locality helps dramatically to prune the search space. SAT solvers BerkMin [4] and Siege [8] have both exhibited great speedups from such decision heuristics.

One way of trying to make VSIDS more local is to increase the frequency of score decay. The variable ordering scheme also differs from the previous version by incrementing the scores of the literals which get resolved out during conflict analysis. Both the submitted versions use a variant of VSIDS where the variables are kept only approximately sorted.

The use of the most recent unsatisfied conflict clause as is done by BerkMin turns out to be a good cost-effective approach to estimate the locality. An unassigned variable with the highest VSIDS literal score in a recent unsatisfied conflict clause is chosen to be branched on. After going through a certain threshold number of conflict clauses, we switch back to the VSIDS decision heuristic.

2.2 Learning Shorter Clauses

Shorter clauses lead to faster BCP and quicker conflict detection. Also, shorter conflict clauses potentially prune larger spaces from the search.

Conflict Driven Clause Learning learns new (conflict) clauses by successively resolving the clauses involved in the current conflict. The sum of the lengths of all the involved clauses usually determines the length of the learned conflict clause, i.e. shorter conflict clauses are more likely to be resolved from a set of short clauses involving a small set of variables.

One simple step to achieve the above goal is to update a variable's antecedent clause with a shorter one whenever possible. This happens many times during BCP, where an already assigned variable gets reassigned with a different antecedent. With almost no additional cost, the old antecedent id can be replaced with the current one if the new antecedent is shorter.

Multiple conflict analysis is a more costly technique. BCP often returns a whole set of conflicting clauses (most of which are derived from some common resolvents). For each conflicting clause, we find the length of the firstUIP clause to be learned, and only add the one with the shortest length. Variables that are assigned at decision level zero are excluded from all the learned conflict clauses.

Conflict clause based assignment stack shrinking [7] is a technique that Jerusat uses and seems to be quite useful in increasing the locality of the search. It is seen that this often reduces the average length of learned conflict clauses. When the firstUIP clause exceeds certain threshold length L, we sort the decision levels of the literals of the clause, backtrack to almost the highest decision level in the clause, and then tell the decision strategy to start reassigning to false the unassigned literals of the conflict clause till a conflict is encountered. We believe this technique works by reducing the size of the set of assigned variables from which future conflict clauses will be derived. In our experiments, no fixed L performed well over a variety of benchmarks. Instead, we set L dynamically using some measured statistics. Zchaff measures the averaged difference between lengths of the clause being used for shrinking and the immediate new clause we get after the shrinking. Zchaff_rand measures the mean and the standard deviation of the lengths of the learned conflict clauses.

2.3 Frequent Restarts

It was found that a fixed interval restart policy was often useful [5] [4].

2.4 Aggressive Clause Deletion

Profiling with `gprof` shows that about 80% of the total running time of zchaff is spent on BCP. Thus long clauses are very harmful when they are not frequently used. We adapt some ideas in BerkMin's clause deletion heuristic. Each clause has its own activity count. This is increased every time the clause is involved in a conflict clause derivation and decreased periodically. We measure the clause's activity to age ratio. Any clause with this ratio less than certain threshold is considered for deletion. The decision to delete the clause is made based on the unrelevance of the clause which is estimated by the number of unassigned literals in the clause.

3 Authors

Zchaff is written in C++. Zchaff was written by Lintao Zhang in 2001 at Princeton University. The current versions have been developed by Yogesh Mahajan and Zhaohui Fu. The official release of zchaff is maintained by Zhaohui Fu and the latest official release can be obtained at `http://www.ee.princeton.edu/~chaff`. We hope to make the latest development versions available soon.

References

[1] `http://www.cs.washington.edu/homes/kautz/blackbox/`.

[2] `http://nusmv.irst.itc.it/`.

[3] `http://www.math.miami.edu/~tptp/ATPSystems/GrAnDe/`.

[4] E.Goldberg and Y.Novikov. Berkmin: a fast and robust SAT-solver. In *DATE*, 2002.

[5] Henry Kautz, Eric Horvitz, Yongshao Ruan, Carla Gomes, and Bart Selman. Dynamic restart policies. In *The Eighteenth National Conference on Artificial Intelligence*, 2002.

[6] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference, Las Vegas*, 2001.

[7] Alexander Nadel. The jerusat SAT solver. Master's thesis, Hebrew University of Jerusalem, 2002.

[8] Lawrence Ryan. The siege satisfiability solver. `http://www.cs.sfu.ca/~loryan/personal/`.