An Experimental Study of Satisfiability Search Heuristics

Fadi A. Aloul
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
faloul@eecs.umich.edu

João P. Marques-Silva
IST/INESC
Cadence European Laboratories
R. Alves Redol, 9
1000 Lisboa, Portugal
jpms@inesc.pt

Karem A. Sakallah
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
karem@eecs.umich.edu

Abstract

Interest in propositional satisfiability (SAT) has been on the rise lately, spurred in part by the recent availability of powerful solvers that are sufficiently efficient and robust to deal with the large-scale SAT problems that typically arise in electronic design automation application. A frequent question that CAD tool developers and users typically ask is which of these various solvers is "best;" the quick answer is, of course, "it depends." In this paper we attempt to gain some insight into, rather than definitively answer, this question.

Introduction. Most modern SAT algorithms can be classified as enhancements to the basic Davis Putnam (DP) backtrack search approach. The DP procedure performs a depth-first search in the *n*-dimensional space of the problem variables and can be viewed as consisting of three main engines: 1) a *decision* engine that makes *elective* assignments to the variables; 2) a *deduction* engine that determines the consequences of these assignments, typically yielding additional *forced* assignments to, i.e. implications of, other variables; and 3) a *diagnosis* engine that handles the occurrence of conflicts (i.e. assignments that cause the formula to become unsatisfiable) and backtracks appropriately. An important distinction that will be critical in explaining performance differences between various SAT algorithms is whether they are *formula* or *function satisfiers*. A formula satisfier performs the search for a satisfying assignment assuming a fixed set of input clauses; it attempts to satisfy a function *f* based on a specific formula φ. The DP algorithm is an example of a formula satisfier. A function satisfier, on the other hand, may modify the formula φ representing the function being satisfied to improve search efficiency. Function satisfiers can be classified based on when and how they modify their input formula. *Static function satisfiers* pre-process the input formula, augmenting it with extra prime implicates and removing from it any subsumed clauses. In contrast, *dynamic function satisfiers* identify "useful" clauses adaptively during the search, either by the deduction or the diagnosis engines. These clauses help generate further implications and may optionally be stored in the clause database for possible future use.

In the context of the DP deduction rules, different formulas representing the same function may possess different "reasoning powers" and may yield vastly different implications. Indeed, it is easy to show that a "complete" formula contains all possible implications to any set of decisions and will not lead to conflicts and backtracking. Unfortunately, a complete formula will generally have an exponential number of clauses, and will likely yield no run time savings. Identifying the "right" formula for a given function, i.e. the formula that consists of just the right number and type clauses to minimize search time, is a clearly desirable but unfortunately unattainable goal.

Experimental Results. We examine the performance of the basic DP search on variants of the input formula that are augmented statically, in a pre-processing step, with additional consensus clauses generated using a truncated iterative consensus procedure. Our experimental results indicate that the addition of consensus clauses to a formula leads to a reduction in the number of decisions and conflicts. On the other hand, the total run time tends to initially decrease then dramatically increase as more clauses are added. Pre-processing necessarily has no knowledge of how the search process will evolve and may create "useless" clauses. On the other hand, dynamic search can augment the clause database with more "useful" clauses than blind pre-processing. We experimentally compare two dynamic function satisfiers: *recursive learning* (RL) which can be viewed as adding "what-if" analysis to the deduction engine to derive more variable assignments following each decision assignment; and *conflict analysis (CD)* which generates implicates related to the occurrence of conflicts, essentially adding "why" analysis to the diagnosis engine. Both RL and CD outperform the DP procedure. Yet, CD was more successful than RL. This may be explained by the fact that RL was too time consuming during the search because it was executed after each decision. In contrast, CD was executed only after each conflict.

Conclusions. The broad conclusions of the study are that supplanting the input formula with more clauses has the potential of significantly reducing the search effort. This gain, however, must be weighed against the extra effort required to generate the additional clauses. Dynamic clause identification, therefore, is generally more effective than a static pre-processing scheme. Furthermore, clauses identified from conflicts seem generally to be more useful in reducing overall search time than those found by RL. A combination of RL and CD may, ultimately, be the most effective approach for dealing with difficult problems.