Search-Based SAT Using Zero-Suppressed BDDs

Fadi A. Aloul, Maher N. Mneimneh, Karem A. Sakallah

Department of EECS, University of Michigan, Ann Arbor 48109-2122

{faloul, maherm, karem}@eecs.umich.edu

Abstract

We introduce a new approach to Boolean satisfiability (SAT) that combines backtrack search techniques and zero-suppressed binary decision diagrams (ZBDDs). This approach implicitly represents SAT instances using ZBDDs, and performs search using an efficient implementation of unit propagation on the ZBDD structure. The adaptation of backtrack search algorithms to such an implicit representation allows for a potential exponential increase in the size of problems that can be handled.

Introduction. Many efficient enhancements to the Davis-Logemann-Loveland (DLL) backtrack-search procedure have been proposed. These enhancements extended the application of SAT solvers to large problem instances. Nevertheless, despite these advances, the tremendous growth in today's designs is outpacing the capabilities of existing SAT solvers as these tend to *explicitly* represent the clause database and lead to time and memory explosion.

Rather than *explicitly* representing the clause database using arrays or linked lists, an alternative is to *implicitly* represent the clause database. Zhang et al. [3] introduced an efficient implementation of the DLL procedure using a trie to store the clauses. Tries allow sharing of nodes among clauses beginning with identical sequences of literals. Recently, Chatalic et al. [1] proposed a new implementation of the *resolution*-based Davis-Putnam procedure using ZBDDs as the underlying data structure for clause encoding. This approach succeeded in solving several SAT instances that defied search-based methods. Unlike tries, ZBDDs allow the sharing of nodes among clauses with common literals regardless of the location of literals in the clauses. The high compression power of this data structure resulted in exponential reductions in space and time complexity for certain instance classes.

More recently, enhancements to the implementation of unit propagation were shown to yield significant performance improvements [2], especially for problems containing large numbers of large clauses. The idea is to keep track of *any two unresolved literals* in each clause as opposed to all literals in the clause database.

In this paper, we present a new *search*-based technique, where the focus is on the data structure used for encoding sets of clauses. ZBDDs are used to implicitly represent the clause database, and unit propagation is implemented by an efficient procedure that processes *sets of, instead of single, clauses*. Our algorithm involves manipulating pointers to ZBDD nodes representing literals that are adjacent in the clause database.

Experimental Results. We used a basic DLL implementation with a fixed variable decision order. We experimentally compared four sets of results corresponding to ZBDDs, tries, and two variants of lists. The ZBDD, trie, and first list variant use two literal pointers per clause; the trie and list implementations replicate SATO and Chaff, respectively. The second list variant replicates the method implemented in GRASP in which a pointer exists for every literal in the instance. A fixed variable order is used for the ZBDD and trie structures. Our benchmarks included various instances from the bounded model checking and DIMACS sets.

Our results showed a correlation between compression rates and search run times. Specifically, ZBDDs provided speedups for instances that were highly compressed (i.e. those that had a large literal count to ZBDD node ratio.) This is easily explained by the fact that our implementation of unit propagation is proportional to ZBDD size rather than total instance size. We also observed that structured instances typically yield higher compression ratios and commensurately higher speedups.

Experimental results do indeed confirm the memory reduction advantages of ZBDDs over lists and tries. Furthermore, reduced memory consumption consistently translates into faster run times despite the initial overhead of building the ZBDD structure.

Conclusions. We described a new implementation of the classic DLL search procedure that uses ZBDDs as the underlying data structure. Compared to explicit lists and tries, ZBDDs have the potential of significantly compressing a clause database leading to faster search times and the possibility of tackling much larger instances. We believe that the incorporation of clause recording, through conflict analysis and recursive learning, will increase the performance edge of ZBDDs over conventional data structures. Furthermore, applying ZBDD variable reordering heuristics can lead to even higher compression ratios, and hence, faster run times.

References

[1] P. Chatalic and L. Simon, "Multi-Resolution on Compressed Sets of Clauses," *in ICTAI*, 2000.

[2] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *in Proc. of the Design Automation Conference*, 2001.

[3] H. Zhang and M. Stickel, "Implementing the Davis-Putnam Algorithm by Tries," *Tech. Report, Univ. of Iowa*, 1994.