Backtrack Search Using ZBDDs

Fadi A. Aloul, Maher N. Mneimneh, Karem A. Sakallah
{faloul, maherm, karem}@umich.edu
Electrical Engineering and Computer Science
University of Michigan

Abstract

We introduce a new approach to satisfiability that combines backtrack search techniques and zero-suppressed binary decision diagrams (ZBDDs). This approach implicitly represents satisfiability problems using ZBDDs, and performs search using operations on this representation. This methodology which adapts backtrack search algorithms to such implicit representations should allow for a potential exponential increase in the size of problems that can be handled. We describe how to perform backtrack search and conflict diagnosis with ZBDDs used as an underlying structure for clause representation. We also report on our initial experiments with this approach.

1 Introduction

Boolean Satisfiability (SAT) serves as an underlying model for a wide range of applications in Computer Science, Artificial Intelligence and Electrical Engineering, to name a few. Over the years, this problem has been extensively investigated and efficient algorithmic solutions eagerly sought. SAT research efforts culminated in an extensive collection of proposed solutions. Of these, the most known *complete* algorithms are based on the Davis-Putnam method [2], and variations of the Davis-Logemann-Loveland method [3]. Despite literature confusion, the two approaches are different. The former, based on *resolution*, performs existential elimination on the propositional variables. The procedure is repeated until the formula equals either 0 (unsatisfiable problem instance) or 1 (satisfiable problem instance). Resolution tends to be memory intensive as existential elimination often generates a large number of clauses. The latter approach, based on *backtrack search*, implicitly enumerates the space of possible binary assignments looking for a satisfying one. A decision tree keeps track of current assignments and prunes the search by iteratively applying *unit propagation*, usually referred to as *Boolean Constraint Propagation* [12]. If a *conflict* is reached, the search backtracks to some previous assignment. *Conflict analysis* [7], and *recursive learning* [5] comprise major enhancements to the basic backtrack search procedure. Conflict analysis comes into play when a conflict arises, and adds adequate information, a *conflict clause*, that anticipates the possible reoccurrence of this conflict. Furthermore, conflict analysis allows the search process to backtrack non-chronologically to earlier levels in the search tree, considerably pruning the search space. On the other hand, recursive learning, when extended to conjunctive normal form (CNF) clauses, identifies necessary assignments by examining the different possible ways of satisfying a given clause from the set of unassigned literals.

These improvements allowed solving large problem instances in various domains [7]. However, search-based approaches are still incapable of handling very large problems arising from various EDA applications [10] as they tend to *explicitly* represent the clause database. This explicit representation and enumeration often results in time and memory explosion.

Recently, this problem has been addressed by Chatalic et al. [1] who proposed implementing resolution using zero-suppressed binary decision diagrams (ZBDDs) [6, 9] as the underlying data structure for clause encoding. Their approach was capable of solving two hard problems that known SAT solvers failed at. The high compression power of the underlying data structure resulted in enormous reductions in algorithmic complexity.

In this paper, we push the above approach further. We explore using such an implicit clause database representation with backtrack search techniques. This is motivated by the desire to integrate the advantages of BDD-based and SAT-based approaches in a hybrid scheme. In addition, we show how to efficiently identify conflicts in such a data structure and how to generate conflict clauses using resolution.

2 Preliminaries

ZBDDs [6, 9] were inspired by the need to efficiently represent and manipulate sets of combinations. It is a directed acyclic graph (DAG) consisting of two terminal nodes, the 0-terminal (the empty set) and the 1-terminal (the set of a single empty combination), and non-terminal nodes each of which has two children, the 1-successor and the 0-successor. In addition, each non-terminal node is labeled with a Boolean variable. Given a universe $U = \{c_1, c_2, \ldots, c_n\}$ of n objects, a combination $C = (c_1, c_2, \ldots, c_m)$ of m objects from U can be represented by an n-bit *binary* vector $X = (x_1, x_2, \ldots, x_n)$ where $x_i = 1$ if object c_i is in C, and 0 otherwise, $1 \le i \le n$. A set S of combinations can be represented by a characteristic function $\chi_S: \{0,1\}^n \rightarrow \{0, 1\}$ where $\chi_S(X) = 1$ if $X \in S$ and 0 otherwise, $X \in \{0, 1\}^n$. In what follows, we use a set S and its characteristic function χ_S interchangeably.

ZBDD node semantics are illustrated in Figure 1(a). If a node v with label x_i represents a set S, and v's 0-successor and 1-successor represent S_0 and S_1 respectively, then $S = S_0 \cup (S_1 \times \{c_i\})$, where:

$$A \times B = \{a \cup b \mid (a \in A) \text{ and } (b \in B)\}. \quad (1)$$

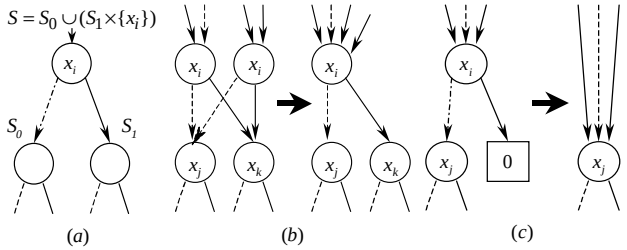

Figure. 1 (a) ZBDD node semantics
(b) node merging rule and (c) node elimination rule

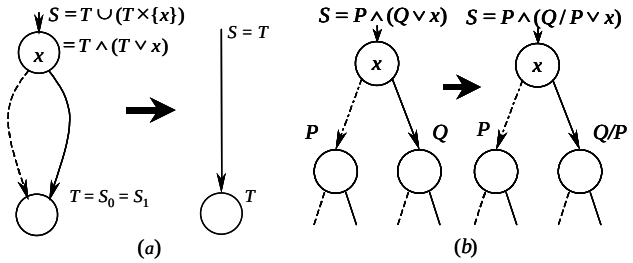

Figure. 2 (a) clause subsumption rule and
(b) subsumption elimination rule

ZBDD construction is based on two reduction rules illustrated in Figure 1(b) and (c). The *node merging rule* merges two nodes if they have the same label and identical 0- and 1-successors, whereas the *node elimination rule* eliminates a node if its 1-successor is the 0-terminal. Each path from the root node to the 1-terminal corresponds to one combination C of S where $x_i = 0$ if no node labeled x_i exists along that path. It is this property that renders ZBDDs a compact representation for sparse combinations. As an example consider the universe $U = \{a, b, c, d, e, f, g\}$ and the set:

$$S = \begin{matrix}\{(a, b, c), (a, b, g), (a, c, d), (e, b, g)\\ (e, c, d), (e, c, h), (f, g, d), (f, g, h)\}\end{matrix} \quad (2)$$

S can be represented by the ZBDD shown in Figure 3(a). (Note that we label the ZBDD nodes with the objects names instead of their encodings for ease of presentation). Minato [9] presented efficient algorithms that implement set theoretic operations on ZBDDs. These operations include *union*, *intersection*, *difference*, and *product*, among others. With efficient caching techniques, these algorithms can execute in time proportional to the ZBDD size rather than the cardinality of the combination sets.

It was demonstrated in [1] that the above approach can be extended to efficiently encode sets of clauses. In this case, each variable and its complement are objects of U, and each path from the root to the 1-terminal corresponds to a single clause. The number of paths to the 1-terminal equals the number of clauses in the clause database. As an example, the set of clauses:

$$S_{cl} = \begin{matrix}(a \vee b \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee c \vee d) \wedge\\ (\neg a \vee b \vee \neg c) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge\\ (\neg b \vee \neg c \vee \neg d) \wedge (\neg b \vee \neg c \vee d)\end{matrix} \quad (3)$$

corresponds to a set of combinations from $U_{cl} = \{a, \neg a, b, \neg b, c, \neg c, d, \neg d\}$ and can be represented by the ZBDD shown in Figure 3(b). Using this approach, the semantics of Boolean Algebra, such as subsumption, can be superimposed on ZBDD reduction rules to achieve further compression. As an example consider the ZBDD illustrated in Figure 2(a) where the 1-successor and the 0-successor of the root are identical. Using ZBDD node semantics, $S = T \wedge (T \vee x)$ and by the subsumption rule of Boolean Algebra, $S = T$. Another CNF-specific reduction rule is *subsumed difference* [1]: given two sets S and T, the subsumed difference of S by T, denoted as S/T, is the set of clauses of S that are not subsumed by any clause from T. In a ZBDD, whose root node represents the set S and its 0 and 1-successors represent P and Q respectively, $S = P \wedge (Q \vee x)$. Since P is independent of x, clauses in P can subsume clauses in $Q \vee x$, while clauses in $Q \vee x$ can't subsume any clause in P. Consequently, $S = P \wedge [(Q \vee x)/P] = P \wedge (Q/P \vee x)$. This reduction rule is illustrated in Figure 2(b). It was shown that the recursive application of this rule results in a ZBDD that is free of subsumed clauses. In addition, subsumed difference can be used as the building block for *subsumption-free union* and *subsumption-free product* operations [1]. These operations were incorporated in a *multi-resolution* version of the DP procedure for satisfiability. Given a clause database φ and the characteristic function χ_φ encoding the clauses of φ, after a variable x is selected for existential elimination, multi-resolution uses standard ZBDD operations to partition the clause set into three sets: φ^0, φ^1 and φ'. φ^0 denotes the set of clauses having the literal x, φ^1 the set of clauses having the literal $\neg x$, and φ' the set of clauses having neither x nor $\neg x$. Existential elimination is performed as follows: $\exists x.\varphi = (\varphi^0|_x \vee \varphi^1|_{\neg x}) \wedge \varphi'$ where $\varphi|_x$ donates the cofactor of φ with respect to x. The union between $\varphi^0|_x$ and $\varphi^1|_{\neg x}$ translates into subsumption-free product on the ZBDDs representing their characteristic functions and the intersection of the result with φ' corresponds to subsumption-free union on the corresponding ZBDDs. The advantage of multi-resolution is the reduction in algorithmic complexity of the operations. Each of the above operations depend on the sizes of the corresponding ZBDDs, measured in number of nodes, and not the size of the clause set (i.e literals) they encode.

3 ZBDDs As a Structure for Backtrack Search

Despite various algorithmic solutions, backtrack search remains the most prevalent technique for attacking the satisfiability problem. Backtrack search implicitly enumerates the space of truth assignments using a decision tree to maintain current assignments to Boolean variables. Although many effective improvements were incorporated in the algorithm, it is still incompetent for large-scale clause databases because of its explicit representation of clauses. To conquer this, we propose an implicit backtrack search algorithm that uses ZBDDs as the underlying data structure for clause rep-

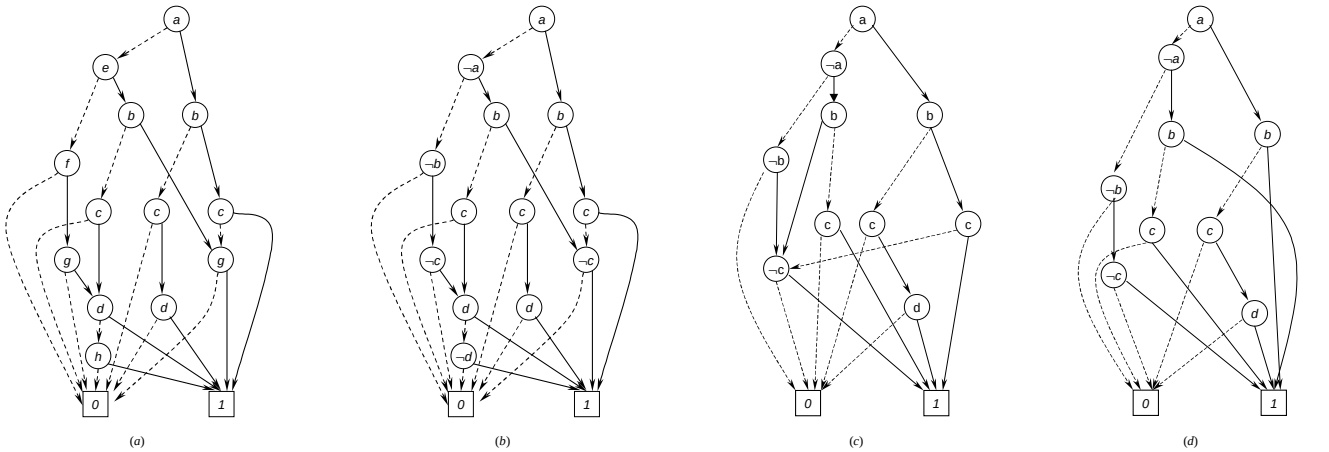

Figure. 3 (a) ZBDD representing S (b) ZBDD representing S_{cl} (c) ZBDD representing S_{cl}^{new} and (d) ZBDD representing S_{cl} after applying reduction rules

resentation.

A generic backtrack search algorithm is comprised of three main engines [8]: *Decide()*, *Deduce()*, and *Diagnose()*. In what follows, we briefly describe each engine and illustrate how to achieve its function when ZBDDs are the underlying data structure.

Decide() uses heuristic knowledge to make *elective* assignments to variables. When using ZBDDs, we assign a variable by adding a one-literal clause, representing this assignment to the clause set. This is achieved by unioning, using subsumption-free union, the characteristic function of the clause to be added with the characteristic function of the clause database. As an example, consider S_{cl} again. To assign a to 1, we add the clause (a) to S_{cl} to get:

$$S_{cl}^{1} = \begin{array}{c}(a) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee c \vee \neg d) \wedge \\ (\neg a \vee c \vee d) \wedge (\neg b \vee \neg c \vee \neg d) \wedge (\neg b \vee \neg c \vee d)\end{array} \quad (4)$$

the three clauses $(a \vee b \vee \neg c)$, $(a \vee b \vee c)$, and $(a \vee c \vee d)$ are subsumed by (a).

Deduce() determines the consequences of the assignments elected by *Decide()*, typically yielding additional *forced* assignments to, i.e. implications of, other variables. This is achieved by repeatedly applying the *unit clause rule* until no unit clauses exist. To implement this approach using ZBDDs, we need to identify unit clauses and absorb on them. Single-literal clauses are identified by recursively traversing "zero-successors" of the ZBDD encoding the clause set starting at the root; any node whose 1-successor is the 1-terminal denotes a unit clause. Absorption is then carried on the identified set of unit clauses by recursively applying the *absorption reduction rule* illustrated in Figure 4. The recursive application of this procedure automatically handles unit propagation, eliminating the need to track implications explicitly. An empty clause, i.e. 1-terminal, designates a conflict, indicating the existence of an unsatisfied clause, while a set consisting of only unit clauses represents a satisfying assignment. Applying absorption to S_{cl}^{1}, we get:

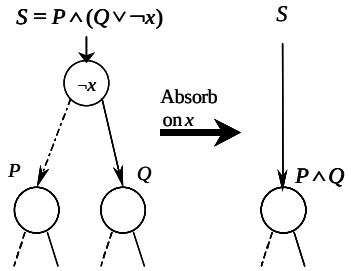

Figure. 4 ZBDD absorption rule

$$S_{cl}^{2} = \begin{array}{c}(a) \wedge (b \vee \neg c) \wedge (c \vee \neg d) \wedge (c \vee d) \wedge \\ (\neg b \vee \neg c \vee \neg d) \wedge (\neg b \vee \neg c \vee d)\end{array} \quad (5)$$

Diagnose() handles the occurrence of conflicts and backtracks appropriately to a previous decision. Besides, it analyzes the causes of the conflict and generates adequate information to prevent its re-occurrence. While our *Deduce()* engine eliminates the need for an implication graph that keeps track of assignments, it lacks the ability to identify variable assignments causing conflicts. To surmount this, we keep a copy of the clause database before each iteration of the unit propagation rule. On a conflict, we use this copy to identify conflicting variables. This is achieved by checking for a pattern of the following structure $x_1 \wedge x_2 \wedge \ldots \wedge x_k \wedge (\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_k)$. The existence of such a pattern indicates that each of x_1 to x_k is a conflicting variable and can now be used to generate the learned clauses, that can effectively prune the search space. We use a novel conflict diagnosis approach that implicitly generates learned clauses by applying resolution on identified conflicting variables. As an example, consider S again. Assume that the current assignments are $a = b = 1$, and the *Decide()* engine selects $c = 1$. This results in $S^{conf} = (d) \wedge (\neg d)$ indicating that d is a conflict variable. Applying resolution on d results in two clauses: $(\neg a \vee c)$ and $(\neg b \vee \neg c)$. These clauses are added to S_{cl} to get:

$$S_{cl}^{new} = \begin{array}{c}(a \vee b \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee c \vee d) \wedge \\ (\neg a \vee b \vee \neg c) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)\end{array} \quad (6)$$

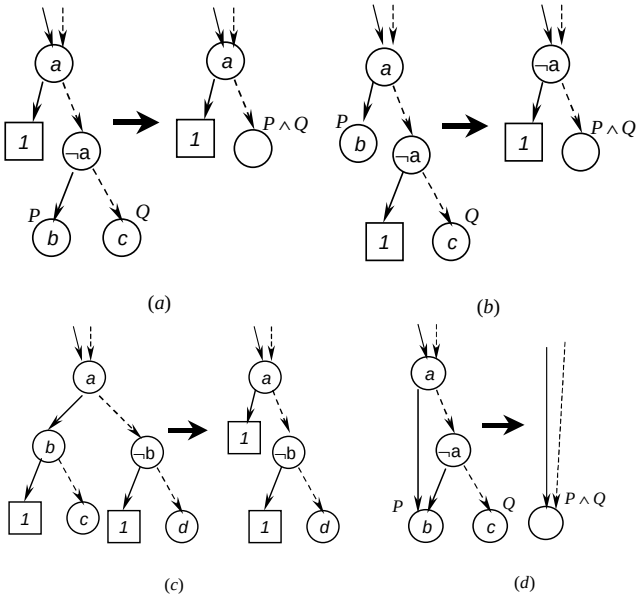

Figure. 5 Boolean algebraic manipulation techniques

Instance	Regular		Reduced		
	Time	Max-CL	Time	Max-CL	Compression
aim-50-3_4-yes1-2	>1000	59492	116.57	9440	5.93
aim-50-1_6-no-1	0.13	84	0.09	69	1.32
aim-200-1_6-yes1-1	>1000	621153	11.08	2520	5.35
jnh2	>1000	123174	>1000	50319	3.60
pret60_25	0.16	600	0.11	164	1.86
dubois22	0.23	6144	0.23	6144	814.55
hole7	22.23	43220	25.09	43220	190.27
ii8a1	0.63	750	0.6	662	7.68
par8-1	17.53	11877	12.38	3967	26.36
pret60_75	0.16	600	0.13	164	1.86
ssa7552-160	117.34	5553	112.98	3094	1.60
hole7	0.3	823697	0.32	823697	2.76E8
hole10	2.07	1E10	2.05	1E10	3.39E4

Table 1: Resolution Results

whose ZBDD is shown in Figure 3(c). The algorithm continues by unioning previous decision i.e, $(a)(b)$ with S_{cl}^{new}.

Since *"knowing more is good"*, augmenting the formula with additional clauses can produce an avalanche of implications during search, consequently leading to a dramatic reduction in search time. However, *"knowing too much is bad"* since applying *"blind"* resolution can generate an exponential number of clauses tremendously slowing down the search performance. Therefore, we impose a limit on the size (number of literals) of the generated clauses as a heuristic for limiting the number of generated clauses. Also, we *selectively* perform resolution on subsets of the clause database that only consists of previously assigned variables. The set of generated clauses are added to the initial clause database using subsumption-free union.

Further *reduction techniques*, illustrated in Figure 5, that perform a combination of Boolean algebraic manipulations (i.e. absorption, subsumption, and resolution) can be locally applied to the ZBDD in order to reduce its size and eventually the number of literals in the problem. Figure 3(d) shows the result of such relations on clause set S_{cl}: whereas the original problem had 8 clauses, 24 literals, and required 13 ZBDD nodes, the reduced formula consisted of 5 clauses, 11 literals, and required only 9 ZBDD nodes.

The time complexity of the described algorithms is a function of the number of nodes in the ZBDD rather than the size of the clause database. With the strong compression power of ZBDDs, this approach promises better results than explicit techniques when dealing with large scale problems.

4 Experimental Results

To check the efficiency of resolution using ZBDDs, we implemented a version of ZRES [1]. Our algorithm is implemented in C++ and uses the CUDD package [11] to build the ZBDDs. Table 1 shows the results for selected problems from the DIMACS set [4]. All experiments were conducted on a Pentium-II 333 MHz machine running Linux and equipped with 512 MB of RAM. The time-out limit was set to 1000 seconds. In general, the resolution approach was inefficient in solving the majority of small benchmarks and unable to solve any of the large benchmarks. With additional reduction techniques, we were able to improve the runtimes but many problems remained unsolved. Table 1 also shows the compression capability, shown in the last column, obtained when using ZBDDs as opposed to explicit lists of clauses. The compression power is measured as the ratio of the number of literal in the problem and the number of nodes in the ZBDD. Clearly, this implicit representation provides great memory reduction, especially on *structured* problems. It is instructive to point out that the *hole-n* results reported in Chatalic et al. [1], shown at the end of Table 1, has a particular structure that can be represented very efficiently using specific ZBDD variable order. In contrast, the results shown in Table 1 were generated using a fixed ZBDD variable order $(1, 2, \ldots, k)$. In general, structured problems suggests a natural efficient variable ordering for ZBDD construction as well as resolution variable elimination.

Figure 6 shows the search runtimes for various size limits of the clauses generated during conflict analysis for four benchmarks. The graphs also show the runtimes for the *regular* and *selective* approach (discussed in Section 3) which applies resolution on the complete problem and on a selected subset of the problem, respectively. The *selective/reduced* curve represents a combination of the *selective* approach with the reduction techniques shown in Figure 5. The data clearly shows the advantage of limiting the number of the generated clauses and their sizes on the performance of the search process. An optimal limit exists for each of the presented benchmarks, which would yield minimal search time.

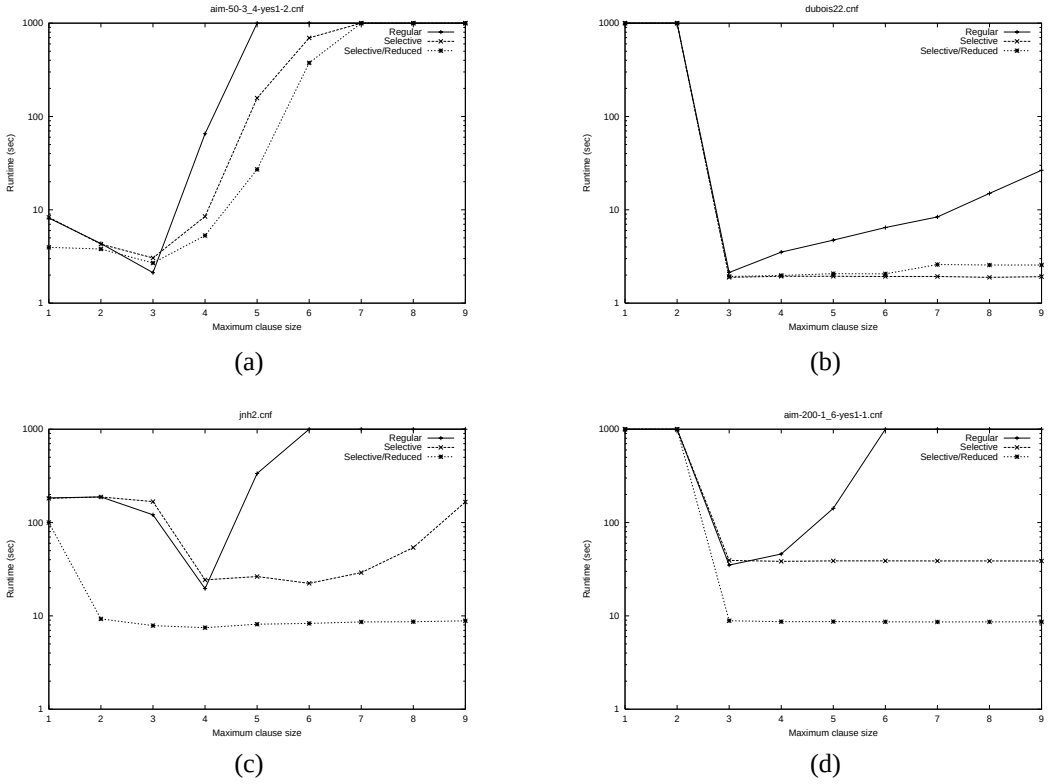

Figure. 6 Backtrack search runtimes using ZBDDs

However, unrestricted clause size limits can lead to an exponential number of clauses, hence severely slowing the search performance. Although the shown benchmarks were easily solved by other SAT solvers, we conjecture that further refinements of the search algorithm, especially conflict diagnosis, should be able to handle very large problems that are out of the scope of current solvers. Such refinements include, efficiently ordering ZBDD variables and dynamically limiting clause size.

5 Conclusion

We have proposed a new approach to satisfiability that combines backtrack search and resolution using ZBDDs. The advantage of such an approach is twofold. Firstly, it promises to be able to deal with large scale clause databases through its implicit representation. Secondly, it serves as a means to study the time-space tradeoff between backtrack search and resolution. Our preliminary results show the effective memory compression achieved with this approach. Further work involves enhancing this technique to handle large problems in feasible time.

6 References

[1] P. Chatalic and L. Simon, "Multi-Resolution on Compressed Sets of Clauses," *in Proc. of the Int'l Conference on Tools with Artificial Intelligence*, 2000.

[2] M. Davis and H. Putnam, "A Computing Procedure Quantification Theory," *in Journal of the ACM*, vol. 7, pp. 201-215, 1960.

[3] M. Davis G. Logemann, D. Loveland, "A Machine Program for Theorem Proving," *in Communications of the ACM*, 1962.

[4] DIMACS Challenge benchmarks in *ftp://Dimacs.rutgers.EDU/pub/challenge/sat/benchmarks/cnf.*

[5] W. Kunz and D. Stoffel, "Reasoning in Boolean Networks," *Kluwer Academic Publishers*, Boston, MA, 1997.

[6] M. Lobbing, O. Schroer, and I. Wegner, "The Theory of Zero-Suppressed BDDs and the Number of Knight's Tours," *in IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, 1995.*

[7] J. P. Marques-Silva and K. Sakallah, "Boolean Satisfiability in Electronic Design Automation," *in Proc. of the Design Automation Conference*, 2000.

[8] J. P. Marques-Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *in IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506-521, 1999.

[9] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *in Proc. of the Design Automation Conference*, 1993.

[10] G. Nam, K. Sakallah, and R. Rutenbar, "Satisfiability-Based Layout Revisited: Detailed Routing of Complex FPGAs Via Search-Based Boolean SAT," *in the Proc. of the Int'l Symposium on Field Programmable Gate Arrays*, 1999.

[11] F. Somenzi, CUDD: CU Decision Diagram Package, University of Colorado at Boulder, *ftp://vlsi.colorado.edu/pub/.*

[12] R. Zabih and D. A. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," *in Proc. of the National Conf. on Artificial Intelligence*, 1998.