Refutation-Complete Binary Decision Diagrams

Scott P. Sanner
Department of Computer Science
University of Toronto
Toronto, ON M5S 3G4
ssanner@cs.toronto.edu

Abstract

Binary decision diagrams (BDDs) are an approach to encoding and reasoning with formulae in propositional logic. They have the property that the representation of any propositional formula is canonical (i.e., two formulae are equivalent *iff* they have the same structure under a variable ordering condition that can be easily enforced); this is particularly useful since tautological and inconsistent formula will reduce to TRUE and FALSE, respectively. However, in traditional theorem proving and satisfiability testing, we typically only need to refute a formula (or equivalently determine that it is unsatisfiable). Since BDDs can both directly prove or refute a formula, they seem to be more powerful than required for many standard applications. Consequently, we analyze the standard BDD algorithms and show that they are performing much more computation than required to provide a refutation. By pruning structure from the BDD that is unnecessary for refutation, we show how the BDD algorithms can be modified to provide an efficient implementation of directional resolution. This yields a refutation-complete BDD (RCBDD) that we implement using CUDD and compare to the original BDD algorithm for satisfiability testing on random 3-SAT formulae. These results show that RCBDDs are an efficient alternative to BDDs for satisfiability testing, with RCBDDs generally outperforming BDDs, and only slightly underperforming BDDs in the remaining cases. The efficiency of RCBDDs and the intuitions developed in this paper make them a promising approach for a wide variety of applications that we discuss in the concluding section.

1 Background

Since the abstract outlines the general structure of the paper, we will immediately delve into the background material required to understand this paper.

1.1 Binary decision diagrams

Binary decision diagrams (BDDs) were introduced in their most well-known form by Bryant [1]. BDDs allow one to encode any propositional formula as a reduced, ordered, directed acyclic decision graph terminating in a TRUE or FALSE terminal node. At each

decision node, the branch corresponding to the variable evaluation is taken (true/false, a.k.a high/low). Once a BDD terminal node is reached, that node represents the evaluation of the formula under the given variable assignment.

To maintain a canonical BDD representation that allows one to determine formula equivalence via graph isomorphism, one must maintain a variable order for all decision nodes and maintain the BDD in reduced form. BDD reduction requires that two properties be maintained:

1. There are only two distinct terminal nodes TRUE and FALSE and no two distinct decision nodes have the same high and low branches (these properties can be maintained via hash tables).
2. Any node having the same child node on its high and low branches is replaced with its child node.

The first rule amounts to an efficient, non-redundant representation for BDDs. We note that this rule is not required to show soundness or completeness of BDD-based theorem proving (where the goal is to directly prove or refute a theorem) but it is required to achieve a canonical graph otherwise. The second rule performs all of the logicaldeduction. For insight into the power of this deduction, see Figure 1. In general we note that the BDD provides an efficient way to store both CNF and DNF formula. In addition, the reduction rules amount to performing refutation resolution for CNF theories and an analogous rule for direct-proof in DNF theories.

1.2 Directional resolution

Directional resolution [2] is the bucket elimination algorithm [3] applied to the Davis-Putnam resolution theorem proving procedure [4]. It essentially amounts to resolving all clauses containing a variable, eliminating that variable, and propagating all remaining clauses to the next variable in the ordering. As for all bucket elimination algorithms, this algorithm has worst-case performance that is exponential in tree-width induced by the variable ordering. An example of directional resolution is given in Figure 2.

2 Refutation-complete BDDs

As noted in Figure 1, the standard BDD algorithms are performing a great deal of unneeded computation if we are only seeking to show unsatisfiability. Specifically, we need not retain the CNF representation of any formula that could not possibly be resolved against in the future. This is the motivation for using variable elimination in the directional resolution algorithm and it can be easily extended to BDDs. We call this extension refutation-complete BDDs (RCBDDs) since they combine the representation of BDDs with the refutation-completeness of directional resolution.

2.1 Algorithm

If we examine Figure 2, we see that we can draw a direct correspondence between directional resolution and BDD algorithms if we note that the CNF formula at each step are represented by the distinct paths to the FALSE terminal node. The basic BDD algorithm already takes care of the resolution rule as noted in Figure 1, so we need only add the variable elimination procedure of directional resolution to this process. We can do this easily by *pruning* out elimated variables and replacing them with TRUE as demonstrated in Figure 2. This pruning step does require extra overhead to reduce the RCBDD to canonical form, but it removes a lot of unnecessary structure that should save a great deal of future computation.

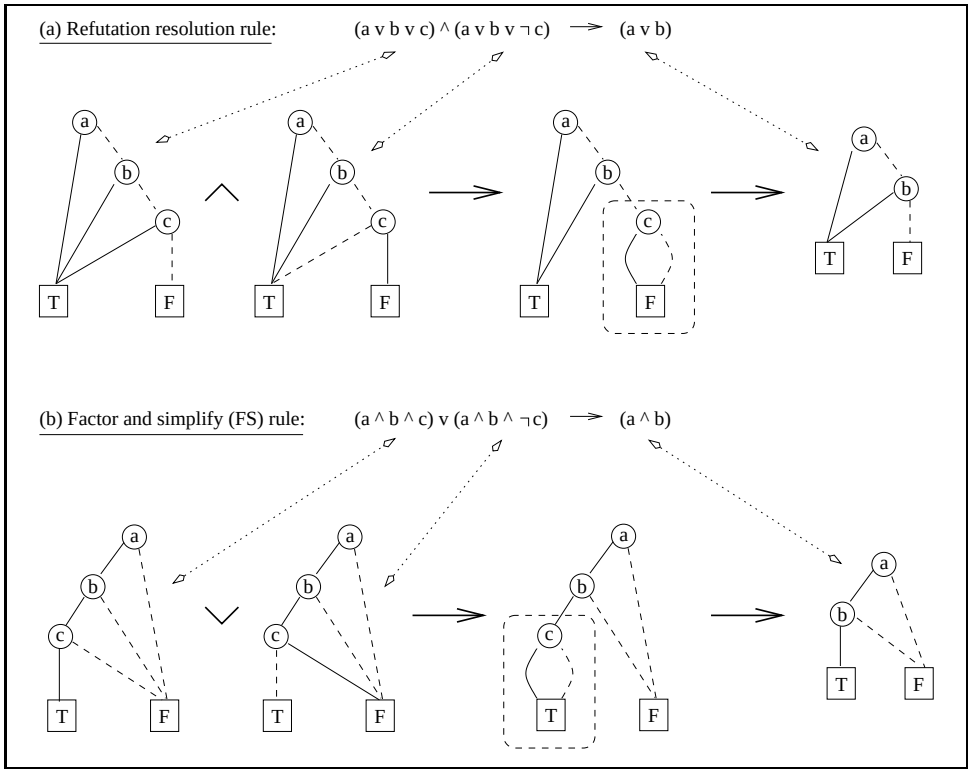


Figure 1: These diagrams depict how the (a) resolution rule for CNF formulae and the (b) factor and simplify rule (FS) rule are implemented for DNF formulae via binary decision diagrams (BDDs). We note that BDDs provide an efficient way of encoding CNF formulae (every path to a F terminal node encodes a CNF clause) and they also provide an efficient way of encoding DNF formulae (every path to a T terminal node encodes a DNF clause). Consequently, given a CNF or DNF clausal representation of a formula, we note that the resulting BDD provides an encoding of this information that efficiently represents redundant structure. Finally, we note that the simplification rules of BDDs effectively implement the resolution rule and the FS rule, thus allowing to both directly prove or refute a propositional theorem. However, we note that directional resolution [2] can achieve refutation completeness using only the resolution rule. Consequently, our refutation-complete approach to BDDs will prune out structure unnecessary for refutation resolution, thereby providing an efficient BDD implementation of directional resolution.

Consequently, we generalize the directional resolution algorithm to BDDs in Figure 3. The correctness of this algorithm follows from the correspondence between BDDs as an efficient CNF representation and the direct correspondence between the RCBDD operations and directional resolution on the CNF formulae.

2.2 Implemention

To implement the RCBDD algorithm in Figure 3, we used the Colorado Universiry decision diagram (CUDD) library developed by Somenzi [5]. This is an extremely efficient C implementation of binary decision diagrams with a number of optimizations.

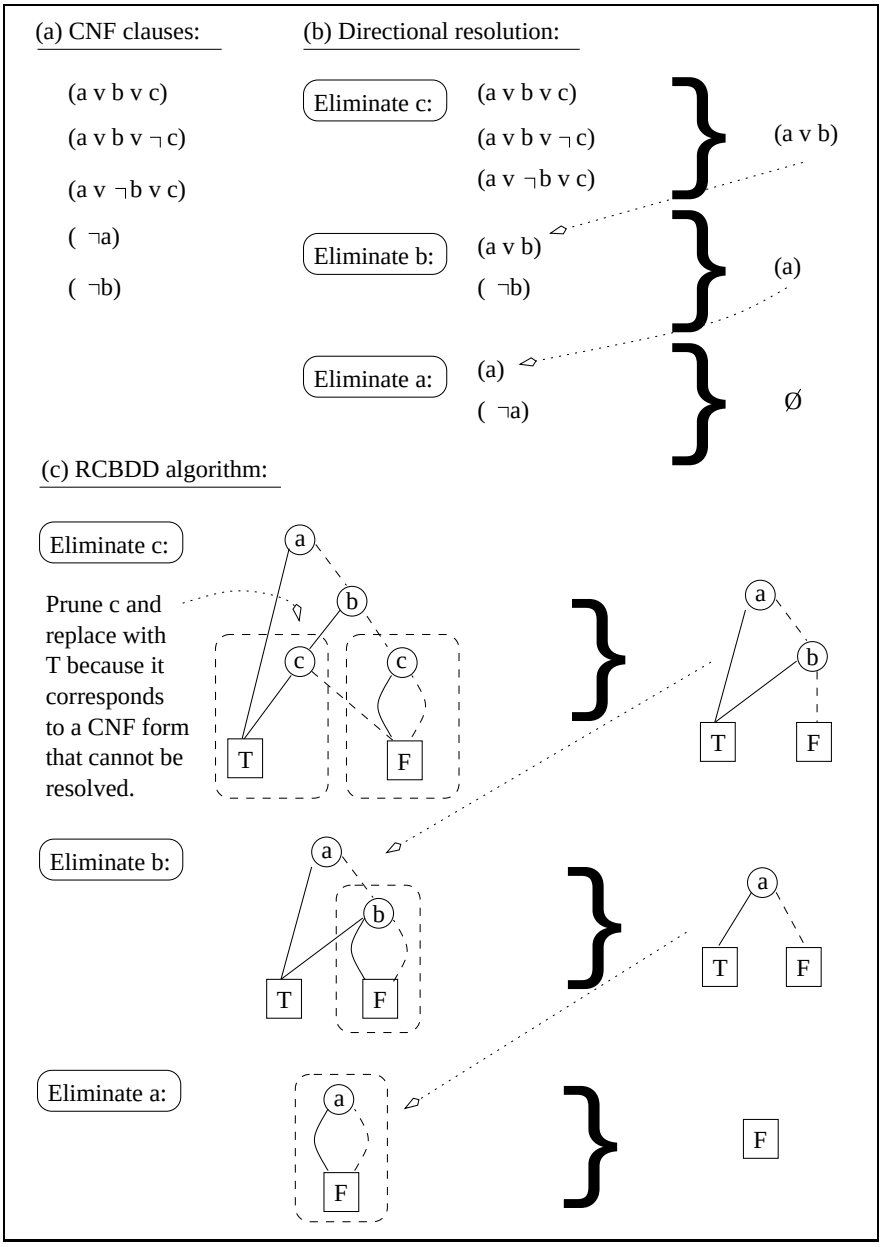


Figure 2: This diagram provides a proof of unsatisfiability for the CNF formulae in (a) via the (b) clausal form of the directional resolution method and (c) the RCBDD form of directional resolution. In directional resolution, the variables are eliminated in order with any resolvents over the eliminated variable passed on to the next step that mentions any of its variables. Clearly the RCBDD is performing resolution over its representation of the CNF formula. The main modification of the general BDD algorithm is the replacement of the eliminated variable with TRUE as demonstrated during the elimination of variable c in part (c). This pruning effectively removes any residual clauses mentioning the eliminated variable from the set (remember, the CNF clauses are implicitly encoded as all distinct paths to FALSE). This procedure is sound and refutation-complete because it is performing exactly the same operation that directional resolution performs when it discards residual clauses during variable elimination.

RCBDD-UNSAT
Input: A set Φ of CNF formulae containing c clauses over n variables.
Output: TRUE if Φ is unsatisfiable, FALSE otherwise.

1. Generate a total variable ordering over the variables $v_1 \ldots v_n$.
2. Generate a bucket b_i for each v_i.
3. For each clause ϕ_j, put it in the bucket corresponding to its variable which comes first in the total order.
4. *for* $i = 1 \ldots n$:
   (a) Build a BDD (standard algorithm) by conjoining all clauses and other BDDs from previous steps in bucket i.
   (b) If the BDD has reduced to the FALSE terminal node, *return* TRUE. (Φ is unsatisfiable.)
   (c) Replace any remaining v_i nodes in the BDD with the terminal FALSE node, and reduce the BDD (standard algorithm).
   (d) Add the BDD to the bucket corresponding to its variable which comes first in the total order (if this is not the final bucket).
5. *return* FALSE. (Φ is satisfiable.))

Figure 3: The algorithm for performing refutation resolution using BDDs. This is essentially an efficient BDD-based version of the directional resolution algorithm given by Dechter and Rish [2].

We implemented the RCBDD algorithm by constructing and manipulating BDDs directly with thethe functions in this library. The only non-standard RCBDD operation was the variable elimination/pruning. We implemented this step somewhat suboptimally by noting that pruning out a variable and replacing it with TRUE is equivalent to rotating it to the top of the variable order and computing the disjunction of its two branches. Performance could likely be enhanced by a more direct manipulation of the decision diagram structure for this pruning.

We have not explored the possibility of finding good variable eliminination orderings for RDBDDs in this paper. However, such work would likely enhance the performance of RCBDDs since our algorithms are inherently limited by the induced tree-width of the variable ordering [2]. For the time being, we use a static chronological variable ordering.

For comparison purposes, we also implemented a standard BDD CNF theorem prover using CUDD. This is simply the RCBDD algorithm without the variable elimination/pruning step.[1]

The implementation is available upon request. It works for any SAT problem specified in DIMACS format and there are a number of scripts provided for data and graph generation.

[1] As a sanity check, we always verified that the original BDD algorithm and the RCBDD algorithm produced the same satisfiability results over all problems. The two different algorithms agreed on all problem instances.

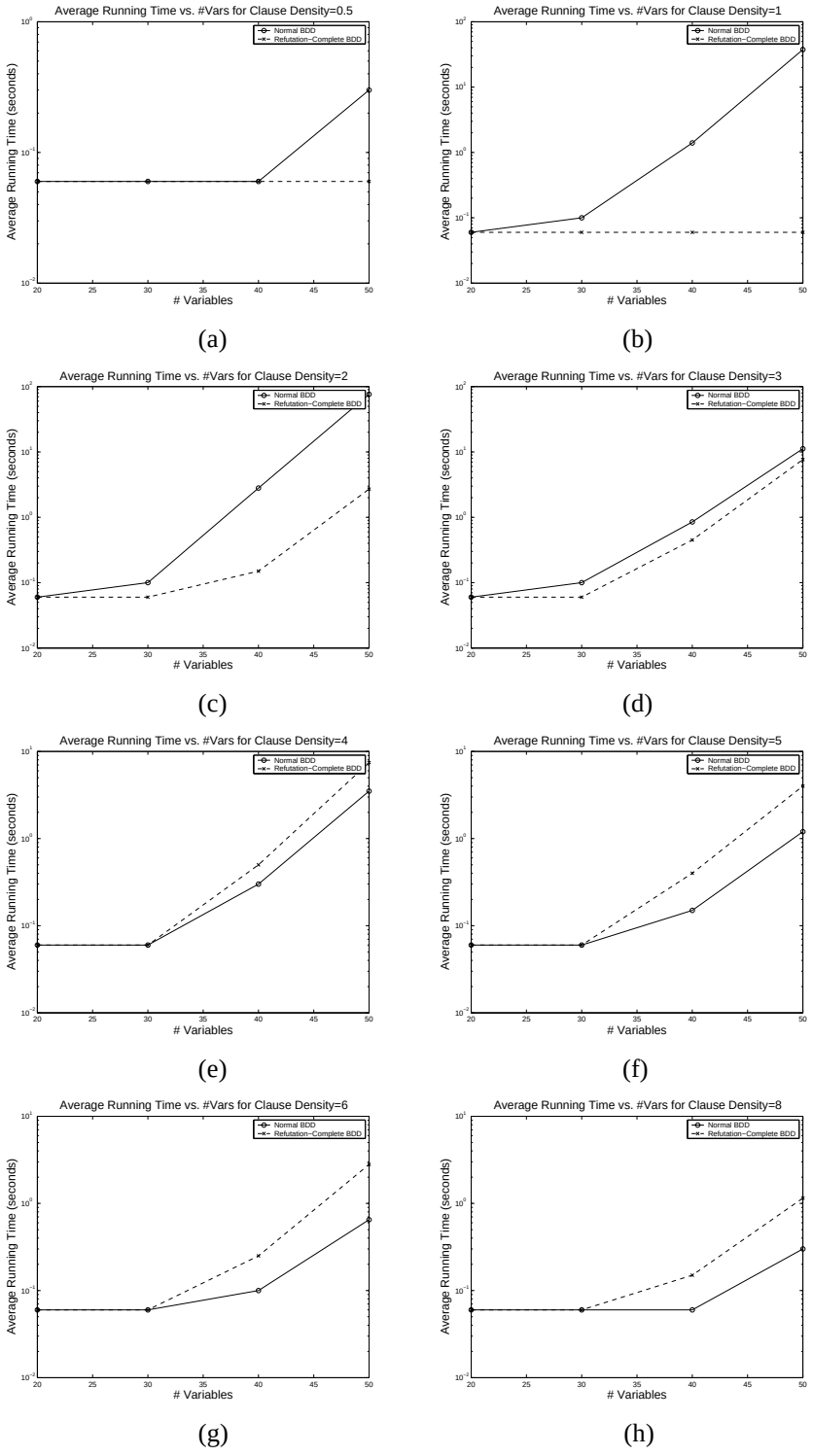
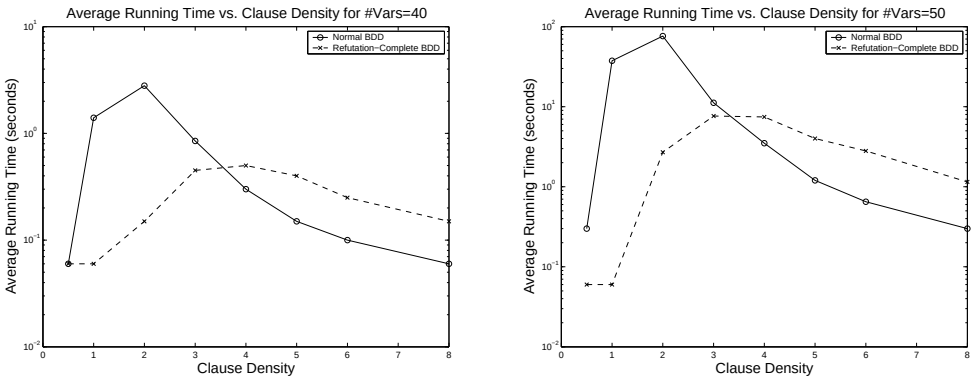


Figure 4: Graphs showing log average running time of satisfiability vs. number of variables (using random 3-SAT instances) for the original BDD algorithm (solid line) and the refutation-complete BDD algorithm (dashed line). Times were averaged over 20 different randomly generated problems. Problems generated for each graph used a fixed clause density ratio (given in the graph title) ranging from 0.5 in (a) to 8 in (h). Both algorithms appear exponential in the number of variables for all density ratios. However, note that for density ratios less than 4, the refutation-complete BDD is more efficient while for density ratios greater than or equal to 4, the normal BDD inference algorithm is more efficient.



(a) (b)

Figure 5: Graphs showing the log average running time of satisfiability vs. clause density ratio (using random 3-SAT instances) for the original BDD algorithm (solid line) and the refutation-complete BDD algorithm (dashed line). Times were averaged over 20 different randomly generated problems. Graph (a) used 40 variable problems and (b) used 50 variable problems. Note that the relative difficulty seems to be directly related to the clause density ratio and is somewhat independent of the number of variables (as expected). As before, the refutation-complete BDD is more efficient for density ratios less than 4 while for density ratios greater than or equal to 4, the normal BDD inference algorithm is more efficient (although not by a large amount). Additionally, it is interesting to note that each algorithm seems to have a different phase transition point (the clause ration where the solution time transitions from increasing to decreasing difficulty for the algorithm). This is explained in the results section.

3 Experiments

3.1 Random 3-SAT

We ran experiments on random 3-SAT instances generated according to the framework discussed in Selman et al. [6]. We used a number of variables in the range $[20, 50]$ and clause to variable ratio (densities) in the range $[0.5, 8]$.

While random 3-SAT instances don't provide a good indicator of real-world performance, they do provide a good comparison test bed where large numbers of problems can be generated over a range of problems that vary in the percentage of satisfiable and unsatisfiable instances as well as their general difficulty. This allows for a comparison of the original and RCBDD algorithms over a wide range of problems.

3.2 Results

Figure 5 shows the average log running time vs. number of variables over a range of clause densities. As noted in the caption, there is a performance switch between for clause densities greater than or equal to 4. At this point, most instances become unsatisfiable and the RCBDD algorithm incurs extra overhead by performing the pruning step when it yields only a few pruned nodes. Consequently, the lighter-weight original BDD algorithm outperforms the RCBDD algorithm on these instances. However, we note that the RCBDD performance is not terrible in comparison to the difference between the BDD and RCBDD algorithms on the satisfiable instances where RCBDD drastically outperforms BDD.

Figure 4 shows the average log running time vs. the clause density for two different variable quantities. As noted in the caption, performance is relatively consistent over different variable quantities. This is expected since the clause density is the primary modulator of problem difficulty. We see that each algorithm has a distinct phase shift point and that futhermore, these are different. Since the RCBDD is reaping the benefits of pruning on the satisfiable instances (low clause density), it is not surprising that the most difficult problems for the RCBDD tend toward the unsatisfiable (where pruning doesn't payoff) while the most difficult problems for the original BDD algorithm occur where a higher percentage of the satisfiable instances are found (since the BDD has a lot of unneeded structure weighing down the algorithm running time).

4 Concluding Remarks

4.1 Achievements

We have introduced a new set of BDD algorithms termed RCBDDs that can be used in place of BDDs for satisfiability-testing (where only refutation-completeness is required). Our empirical results show that RCBDDs are an efficient alternative to BDDs for satisfiability testing, with RCBDDs generally outperforming BDDs, and only slightly underperforming BDDs in the remaining cases.

4.2 Future work

In presenting this work, we have omitted one major detail. The recent development of DPLL-style [7] backtracking algorithms augmented with techniques for clause learning such as Chaff [8] and BerkMin [9] have allowed them to advance far beyond the state-of-the-art performance of BDDs for satisfiability testing. While this means that binary decision diagrams (BDDs) are rarely used for general satisfiability testing, there are nevertheless a variety of special cases where BDDs are the algorithms of choice. The presence of such application domains holds the possibility that the RCBDD algorithms may be applied to enhance BDD performance in these areas. And additionally, the fact that BDDs are rarely used for satisfiability testing does not preclude the possibility that they cannot be extended or combined with other algorithms in ways that overcome their current limitations.

Some of these potential applications are discussed next.

4.2.1 Specialized problem structure

Uribe and Stickel [10] discuss specialized problems such as functional equivalence and domains with heavy recursive structure where BDDs typically perform well. Given the advantages of using BDDs in these domains, it is likely that RCBDDs could enhance performance on these problems.

4.2.2 Finite-state automata verification

Problems such as verifying properties of finite-state automata [11] cannot be used with DPLL algorithms since these algorithms typically have to handle infinite numbers of state transitions. This can be done efficiently by representing transitions as BDDs and computing reachability using fixed-point detection and BDD operations. In this case, it may be possible to apply RCBDDs by regressing faulty states through the transition diagram and using RCBDD operations to infer unsatisfiability of predecessor states, similar to a BDD approach described by Gupta et al. [12]. Using RCBDDs in place of BDDs may lead to a more efficient implementation of property verification in finite-state automata.

4.2.3 Integration with DPLL clause learning

Many of the reasons for the advanced performance of recent DPLL clause learning algorithms stem from the fact that clauses learned from conflicts act as no-goods and allow for efficient conflict detection and backtracking in DPLL search. In some cases, this clause learning can yield exponential speedup in DPLL performance [13]. Since one must obviously store these learned clauses in some data structure, it might make sense to store them in a BDD since this allows for both efficient CNF/no-good storage (based on our previous observations) and efficient conflict detection (just traverse the BDD for the current assignment). Furthermore, an RCBDD could be used to perform directional resolution on these learned clauses, thus increasing the power of clause learning. And when backtracking beyond a variable in the currently learned clauses, one can easily use the previous RCBDD pruning procedure for efficiently simplifying the clause set. Consequently, there are a number of exciting possibilities for RCBDD storage and inference with learned clauses that could enhance both the efficiency and reasoning power of the DPLL algorithms.

References

[1] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[2] Rina Dechter and Irina Rish. Directional resolution: The Davis-Putnam procedure, revisited. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pages 134–145. Morgan Kaufmann, San Francisco, California, 1994.

[3] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In E. Horvitz and F. Jensen, editors, *Twelthth Conf. on Uncertainty in Artificial Intelligence*, pages 211–219, Portland, Oregon, 1996.

[4] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[5] F. Somenzi. Cudd: Cu decision diagram package release, 1998.

[6] Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[7] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, pages 394–397, 1962.

[8] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[9] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver. In *Design Automation and Test in Europe (DATE)*, 2002.

[10] T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In J. P. Jouannaud, editor, *1st International Conference on Constraints in Computational Logics*, volume 845 of *LNCS*, pages 34–49. Springer-Verlag Inc, September 1994.

[11] Guoqiang Pan, Ulrike Sattler, and Moshe Y. Vardi. Bdd-based decision procedures for k. In *CADE-18*, 2002.

[12] Aarti Gupta, Malay K. Ganai, Chao Wang, Zijiang Yang, and Pranav Ashar. Abstract and BDDs complement sat-based BMC in DiVer. In *Computer Aided Verification, 15th International Conference*, volume 2725 of *Lecture Notes in Computer Science*. Springer, 2003.

[13] Paul Beame, Henry Kautz, , and Ashish Sabharwal. Understanding the power of clause-driven learning. In *In Proceedings of the 18th International Joint Conference in Artificial Intelligence IJCAI*, pages 94–99, 2003.