Symbolic Techniques in Satisfiability Solving *

Guoqiang Pan and Moshe Y. Vardi
Department of Computer Science, Rice University, Houston, USA
({gqpan,vardi}@cs.rice.edu*)*

Abstract. Recent work has shown how to use BDDs for satisfiability solving. The idea of this approach, which we call *symbolic quantifier elimination*, is to view an instance of propositional satisfiability as an existentially quantified propositional formula. Satisfiability solving then amounts to quantifier elimination; once all quantifiers have been eliminated we are left with either **1** or **0**. Our goal in this work is to study the effectiveness of symbolic quantifier elimination as an approach to satisfiability solving. To that end, we conduct a direct comparison with the DPLL-based ZChaff, as well as evaluate a variety of optimization techniques for the symbolic approach. In comparing the symbolic approach to ZChaff, we evaluate scalability across a variety of classes of formulas. We find that no approach dominates across all classes. While ZChaff dominates for many classes of formulas, the symbolic approach is superior for other classes of formulas.

Once we have demonstrated the viability of the symbolic approach, we focus on optimization techniques for this approach. We study techniques from constraint satisfaction for finding a good plan for performing the symbolic operations of conjunction and of existential quantification. We also study various variable-ordering heuristics, finding that while no heuristic seems to dominate across all classes of formulas, the maximum-cardinality search heuristic seems to offer the best overall performance.

Keywords: Satisfiability, Binary Decision Diagram, Symbolic Decision Procedure

1. Introduction

Propositional-satisfiability solving has been an active area of research throughout the last 40 years, starting from the resolution-based algorithm in [24] and the search-based algorithm in [23]. The latter approach, referred to as the DPLL approach, has since been the method of choice for satisfiability solving. In the last ten years, much progress has been made in developing highly optimized DPLL solvers, leading to efficient solvers such as ZChaff [45] and BerkMin [33], all of which use advanced heuristics in choosing variable splitting order, in performing efficient Boolean constraint propagation, and in conflict-driven learning to prune unnecessary search branches. These solvers are so effective that they are used as generic problem solvers, where problems such as

* A preliminary version of the paper was presented in SAT'04. Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, ANI-0216467, and by BSF grant 9800096.

bounded model checking [8], planning [39], scheduling [20], and many others are typically solved by reducing them to satisfiability problems.

Another successful approach to propositional reasoning is that of decision diagrams, which are used to represent propositional functions. An instance of the approach is that of ordered Boolean decision diagrams (BDDs) [12], which are used successfully in model checking [14] and planning [17]. A BDD representation also enables easy satisfiability checking, which amounts to deciding whether it is different than the empty BDD [12]. Since decision diagrams usually represent the set of all satisfying truth assignments, they incur a significant overhead over search techniques that focus on finding a single satisfying assignment [19]. Thus, published comparisons between search and BDD techniques [40, 55] used search to enumerate all satisfying assignments. The conclusion of that comparison is that no approach dominates; for certain classes of formulas search is superior, and for other classes of formulas BDDs are superior.

Recent work has shown how to use BDDs for satisfiability solving rather than enumeration [50]. The idea of this approach, which we call *symbolic quantifier elimination*, is to view an instance of propositional satisfiability as an existentially quantified propositional formula. Satisfiability solving then amounts to quantifier elimination; once all quantifiers have been eliminated we are left with either **1** or **0**. This enables us to apply ideas about existential quantifier elimination from model checking [49] and constraint satisfaction [26]. The focus in [50] is on expected behavior on random instances of 3-SAT rather than on efficiency. In particular, only a minimal effort is made to optimize the approach and no comparison to search methods is reported. Nevertheless, the results in [50] show that BDD-based algorithms behave quite differently than search-based algorithms, which makes them worthy of further investigation. (Other recent approaches reported using decision diagrams in satisfiability solving [15, 22, 29, 46]. We discuss these works later).

Our goal in this paper is to study the effectiveness of symbolic quantifier elimination as an approach to satisfiability solving. To that end, we conduct a direct comparison with the DPLL-based ZChaff, as well as evaluate a variety of optimization techniques for the symbolic approach. In comparing the symbolic approach to ZChaff we use a variety of classes of formulas. Unlike, however, the standard practice of comparing solver performance on benchmark suites [42], we focus here on *scalability*. That is, we focus on scalable classes of formulas and evaluate how performance scales with formula size. As in [55] we find that no approach dominates across all classes. While ZChaff dominates

for many classes of formulas, the symbolic approach is superior for other classes of formulas.

Once we have demonstrated the viability of the symbolic approach, we focus on optimization techniques. The key idea underlying [50] is that evaluating an existentially quantified propositional formula in conjunctive-normal form requires performing several instances of conjunction and of existential quantification. The goal is to find a good plan for these operations. We study two approaches to this problem. The first is Bouquet's method (BM) of [50] and the second is the *bucket-elimination* (BE) approach of [26]. BE aims at reducing the size of the support set of the generated BDDs through quantifier elimination and it has the theoretical advantage of being, in principle, able to attain optimal support set size, which is the *treewidth* of the input formula [28]. Nevertheless, we find that for certain classes of formulas BM is superior to BE.

The key to good performance in both BM and BE is in choosing a good variable order for quantification and BDD order. Finding an optimal order is by itself a difficult problem (computing the treewidth of a given graph is NP-hard [4]), so one has to resort to various heuristics, cf. [41]. No heuristic seems to dominate across all classes of formulas, but the maximal-cardinality-search (MCS) heuristic seems to offer the best overall performance.

Finally, we contrast our symbolic solvers with two other solvers, using the MCS variable order. We re-implemented ZRes, the ZDD-based multi-resolution approach of [15], and ZChaff, the DPLL-based solver of [45] to use the MCS variable order. The goal is to have a comparison of the different techniques, using the same variable order. See further discussion below.

We start the paper with a description of symbolic quantifier elimination as well as the BM approach in Section 2. We then describe the experimental setup in Section 3. In Section 4 we compare ZChaff with BM and show that no approach dominates across all classes of formulas. In Section 5 we compare BM with BE and study the impact of various variable-ordering heuristics. In Section 6 we compare our BDD-based algorithm with A ZDD-based algorithm based on ZRes[15] and compare the dynamic variable decision order used in ZChaff with a structural-guided static variable order. We conclude with a discussion in Section 7.

2. Background

A binary decision diagram (BDD) is a rooted directed acyclic graph that has only two terminal nodes labeled **0** and **1**. Every non-terminal node is labeled with a Boolean variable and has two outgoing edges labeled 0 and 1. An ordered binary decision diagram (BDD) is a BDD with the constraint that the input variables are ordered and every path in the BDD visits the variables in ascending order. We assume that all BDDs are *reduced*, which means that where every node represents a distinct logic function. BDDs constitute an efficient way to represent and manipulate Boolean functions [12], in particular, for a given variable order, BDDs offer a canonical representation. Checking whether a BDD is satisfiable is also easy; it requires checking that it differs from the predefined constant **0** (the empty BDD). We used the CUDD package for managing BDDs [53]. The *support set* of a BDD is the set of variables labeling its internal nodes.

In [19, 55], BDDs are used to construct a compact representation of the set of all satisfying truth assignments of CNF formulas. The input formula φ is a conjunction $c_1 \wedge \ldots \wedge c_m$ of clauses. The algorithm constructs a BDD A_i for each clause c_i. (Since a clause excludes only one assignments to its variables, A_i is of linear size.) A BDD for the set of satisfying truth assignment is then constructed incrementally; B_1 is A_1, while B_{i+1} is the result of $\textsc{apply}(B_i, A_i, \wedge)$, where $\textsc{apply}(A, B, \circ)$ is the result of applying a Boolean operator $\circ$ to two BDDs A and B. Finally, the resulting BDD B_m represents all satisfying assignments of the input formula.

We can apply existential quantification to a BDD B:

$$(\exists x)B = \textsc{apply}(B|_{x\leftarrow 1}, B|_{x\leftarrow 0}, \vee),$$

where $B|_{x\leftarrow c}$ restricts B to truth assignments that assign the value c to the variable x. Note that quantifying x existentially eliminates it from the support set of B. The satisfiability problem is to determine whether a given formula $c_1 \wedge \ldots \wedge c_m$ is satisfiable. In other words, the problem is to determine whether the existential formula $(\exists x_1)\ldots(\exists x_n)(c_1 \wedge \ldots \wedge c_m)$ is true. Since checking whether the final BDD B_m is equal to **0** can be done by CUDD in constant time, it makes little sense, however, to apply existential quantification to B_m. Suppose, however, that a variable x_j does not occur in the clauses $c_{i+1}, \ldots, c_m$. Then the existential formula can be rewritten as

$$(\exists x_1)\ldots(\exists x_{j-1})(\exists x_{j+1})\ldots(\exists x_n)((\exists x_j)(c_1 \wedge \ldots \wedge c_i) \wedge (c_{i+1} \wedge \ldots \wedge c_m)).$$

Pursuing this rewriting strategy as aggressively as possible, we process the clauses in the order $c_1, \ldots, c_n$, quantifing variables existentially as

soon as possible (that is, a variable is quantified as soon as it does not occur anymore in the unprocesses clauses). We refer to this as *early quantification* of variables. Note that different clause orders may induce different orders of variable quantification. Finding a good clause order is a major focus of this paper.

This motivates the following change in the earlier BDD-based satisfiability-solving algorithm [50]: after constructing the BDD B_i, quantify existentially variables that do not occur in the clauses $c_{i+1}, \ldots, c_m$. In this case we say that the quantifier $\exists x$ has been *eliminated*. The computational advantage of quantifier elimination stems from the fact that reducing the size of the support set of a BDD typically (though not necessarily) results in a reduction of its size; that is, the size of $(\exists x)B$ is typically smaller than that of B. In a nutshell, this method, which we describe as *symbolic quantifier elimination*, eliminates all quantifiers until we are left with the constant BDD 1 or 0. Symbolic quantifier elimination was first applied to SAT solving in [34] (under the name of *hiding functions*) and tried on random 3-SAT instances. The work in [50] studied this method further, and considered various optimizations. The main interest there, however, is in the behavior of the method on random 3-SAT instances, rather in its comparison to DPLL-based methods.[1]

So far we processed the clauses of the input formula in a linear fashion. Since the main point of quantifier elimination is to eliminate variables as early as possible, reordering the clauses may enable us to do more aggressive quantification. That is, instead of processing the clauses in the order $c_1, \ldots, c_m$, we can apply a permutation π and process the clauses in the order $c_{\pi(1)}, \ldots, c_{\pi(m)}$. The permutation π should be chosen so as to minimize the number of variables in the support sets of the intermediates BDDs. This observation was first made in the context of symbolic model checking, cf. [13, 9, 32, 36]. Unfortunately, finding an optimal permutation π is by itself a difficult optimization problem, motivating heuristic approaches.

A particular heuristic that was proposed in the context of symbolic model checking in [49] is that of *clustering*. In this approach, the clauses are not processed one at a time, but several clauses are first partitioned into several clusters. For each cluster C we first apply conjunction to all the BDDs of the clauses in the C to obtain a BDD B_C. The clusters are then combined, together with quantifier elimination, as described earlier. Heuristics are required both for clustering the clauses and ordering the clusters. Bouquet proposed the following heuristic in [11] (the focus

[1] Note that symbolic quantifier elimination provides *pure* satisfiability solving; the algorithm returns 0 or 1. To find a satisfying truth assignment when the formula is satisfiable, the technique of self-reducibility can be used, cf. [5].

there is on enumerating prime implicants). Consider some order of the variables. Let the *rank* (from 1 to n) of a variable x be $rank(x)$, let the rank $rank(\ell)$ of a literal ℓ be the rank of its underlying variable, and let the rank $rank(c)$ of a clause c be the maximum rank of its literals. The clusters are the equivalence classes of the relation $\sim$ defined by: $c \sim c'$ iff $rank(c) = rank(c')$. The rank of a cluster is the rank of its clauses. The clusters are then ordered according to increasing rank. For example, given the set of clauses $\{x_1 \vee \neg x_2, x_1 \vee x_3, \neg x_2 \vee x_3, x_3 \vee x_4\}$, The clusters are $C_1 = \{\}, C_2 = \{x_1 \vee \neg x_2\}, C_3 = \{x_1 \vee x_3, \neg x_2 \vee x_3\}, C_4 = \{x_3 \vee x_4\}$.

Satisfiability solving using symbolic quantifier elimination is a combination of clustering and early quantification. We keep a set of *active* variables as we conjoin clusters, in the order $C_1, \ldots, C_n$. Starting from an empty set, after each cluster C_i is processed, we add all the variables that occur in C_i to the active set. Then, a variable that does not occur in all C_js where $j > i$ can be removed from the active set and eliminated via early quantification. So, we are computing $\exists X_n \ldots (\exists X_2(((\exists X_1)C_1) \wedge C_2) \ldots \wedge C_n)$, where the quantified variable set X_i consists of the active variables that can be quantified early after C_i is processes. (CUDD allows quantifying several variables in function call.) When we use Bouquet's clustering, the method is referred to in [50] as *Bouquet's Method*, which we abbreviate here is as BM. For the example above, the BM quantification schedule is $\exists x_3 x_4((\exists x_1 x_2((C_1 \wedge C_2) \wedge C_3)) \wedge C_4)$.

We still have to chose a variable order. An order that is often used in constraint satisfaction [25] is the "maximum cardinality search" (MCS) order of [54], which is based on the graph-theoretic structure of the formula. The graph associated with a CNF formula $\varphi = \bigwedge_i c_i$ is $G_\varphi = (V, E)$, where V is the set of variables in φ and an edge $\{x_i, x_j\}$ is in E if there exists a clause c_k such that x_i and x_j occur in c_k. We refer to G_φ as the *Gaifman graph* of φ. MCS ranks the vertices from 1 to n in the following way: as the next vertex to rank, select the vertex adjacent to the largest number of previously ranked vertices (ties can be broken in various ways). The variable order used for the BDDs in the comparisons unless otherwise mentioned is the inverse of the MCS order[2] (See Section 5.2 for exceptions).

[2] Using the MCS order or its inverse as the BDD variable order exhibits little performance difference, so the inverse is preferred because the BE approach presented in Section 5.1 is easier to implement on the inverse order.

3. Experimental setup

We compare symbolic quantifier elimination to ZChaff across a variety of classes of formulas. Unlike the standard practice of comparing solver performance on benchmark suites [42], our focus here is not on simple time comparison, but rather on *scalability*. That is, we focus on scalable classes of formulas and evaluate how performance *scales* with formula size. We are interested in seeing which method scales better, i.e., polynomial vs. exponential scalability, or different degrees of exponential scalability. Our test suite includes both random and nonrandom formulas (for random formulas we took 60 samples per case and reported median time). Experiments were performed using x86 emulation on the Rice Terascale Cluster[3], which is a large Linux cluster of Itanium II processors with 4GB of memory each.

Our test suite includes the following classes of formulas:

- Random 3-CNF: We chose uniformly k 3-clauses over n variables. The *density* of an instance is defined as k/n. We generate instances at densities 1.5, 6, 10, and 15, with up to 200 variables, to allow comparison for both under-constrained and over-constrained cases. (It is known that the satisfiability threshold of such formulas is around 4.25 [52]).

- Random affine 3-CNF: Affine 3-CNF formulas belongs to a polynomial class as classified by Schaefer [51]. Here, they are generated in the same way as random 3-CNF formulas, except that the constraints are not 3-clauses, but parity equations in the form of $l_1 \oplus l_2 \oplus l_3 = 1$[4]. Each constraint is then converted into four clauses $l_1 \vee l_2 \vee l_3, \neg l_1 \vee \neg l_2 \vee l_3, \neg l_1 \vee l_2 \vee \neg l_3, l_1 \vee \neg l_2 \vee \neg l_3$, yielding CNF formulas. The satisfiability threshold of such formula is found empirically to be around density (number of equations divided by number of variables) 0.95. We generate instances of density 0.5 and 1.5, with up to 400 variables.

- Random biconditionals: Biconditional formulas, also known as Urquhart formulas, form a class of affine formulas that have provably exponential resolution proofs. A biconditional formula has the form $l_1 \leftrightarrow (l_2 \leftrightarrow (\ldots (l_{k-1} \leftrightarrow l_k) \ldots))$, where each l_i is a positive literal. Such a formula is valid if either all variables occur an even number of times or all variables occur an odd number of times [56].

[3] `http://www.citi.rice.edu/rtc/`

[4] This is equivalent to just choosing three variables and generate $x_1 \oplus x_2 \oplus x_3 = p$ where $p = 0$ or $p = 1$ with equal probability.

We generate valid formulas with up to 100 variables, where each variable occurs 3 times on average.

– Random chains: The classes described so far all have an essentially uniform random Gaifman graph, with no underlying structure. To extend our comparison to structured formulas, we generate random chains [27]. In a random chain, we form a long chain of random 3-CNF formulas, called *subtheories*. (The chain structure is reminiscent to the structure typically seen in satisfiability instances obtained from bounded model checking [8] and planning [39].) We use a similar generation parameters as in [27], where there are 5 variables per sub-theory and 5-23 clauses per sub-theory, but that we generate instances with a much bigger number of sub-theories, scaling up to > 20000 variables and > 4000 sub-theories.

– Nonrandom formulas: As in [55], we considered a variety of formulas with very specific scalable structure:

  - The n-Rooks problem (satisfiable).
  - The n-Queens problem (satisfiable for $n > 3$).
  - The pigeon-hole problem with $n + 1$ pigeons and n holes (unsatisfiable).
  - The mutilated-checkerboard problem, where an $n \times n$ board with two diagonal corner tiles removed is to be tiled with 1×2 tiles (unsatisfiable).

4. Symbolic vs. search approaches

Our goal in this section is to address the viability of symbolic quantifier elimination. To that end we compare the performance of BM against ZChaff[5], a leading DPLL-based solver across the classes of formulas described above, with a focus on scalability. For now, we use the MCS variable order.

In Figure 1A and 1B, we can see that BM is not very competitive for random 3-CNF formulas. At density 1.5, ZChaff scales polynomially, while BM scales exponentially. At density 6.0 and at higher densities, both methods scale exponentially, but ZChaff scales exponentially better. (Note that above density 6.0 both methods scale better as the density increases. This is consistent with the experimental results in [19] and [50].) A similar pattern emerges for random affine formulas,

[5] ZChaff version 2004.5.13

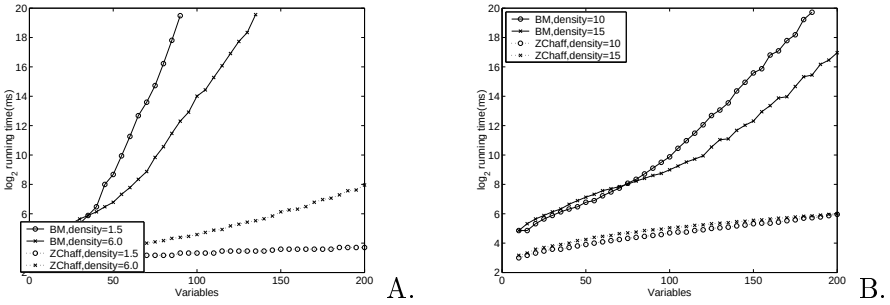


Figure 1. Random 3-CNF

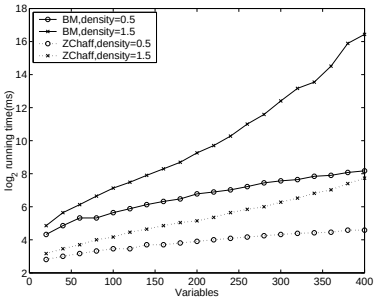


Figure 2. Random 3-Affine

see Figure 2. Again, ZChaff scales exponentially better than BM. (Note that both methods scale exponentially at the higher density, while it is known that affine satisfiability can be determined in polytime using Gaussian elimination [51].)

The picture changes for biconditional formulas, as shown in Figure 3A. Again, both methods are exponential, but BM scales exponentially better than ZChaff. (This result is consistent with the finding in [15], which compares search-based methods to ZDD-based multiresolution.)

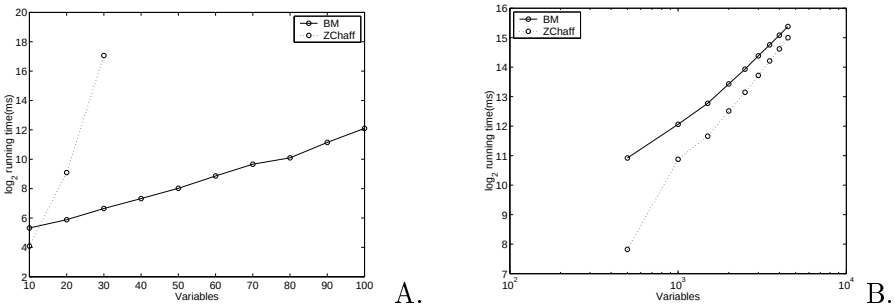


Figure 3. A) Random Biconditionals B) Random Chains

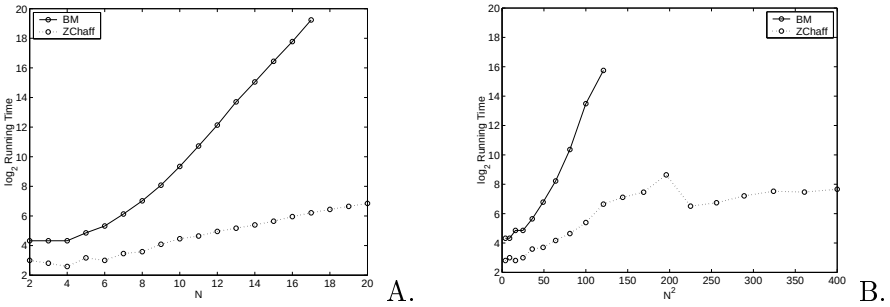


Figure 4. A) n-Rooks B) n-Queens

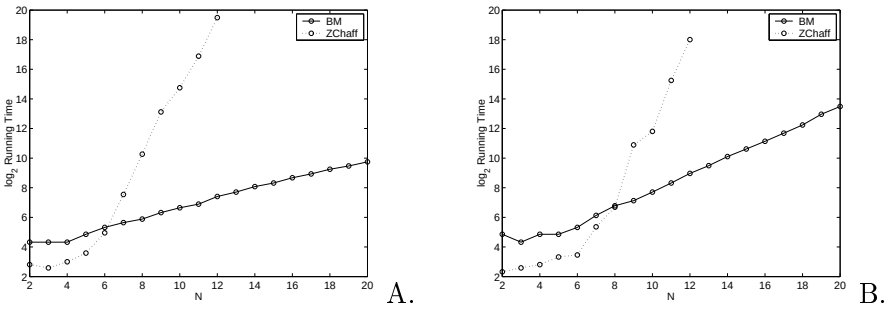


Figure 5. A) Pigeon Hole B) Mutilated Checkerboard

For random chains, see Figure 3B, which uses a log-log scale. Both methods scale polynomially on random chains. (Because density for the most difficult problems change as the size of the chains scales, we selected here the hardest density for each problem size.) Here BM scales polynomially better than ZChaff. Note that for smaller instances ZChaff outperforms BM, which justifies our focus on scalability rather than on straightforward benchmarking.

Finally, we compare BM with ZChaff on the non-random formulas of [55]. The n-Rooks problem is a simpler version of n-Queens problem, where the diagonal constraints are not used. For n-Rooks, the results are as in Figure 4A. This problem have the property of being *globally consistent*, i.e., any consistent partial solution can be extended to a solution [25]. Thus, the problem is trivial for search-based solvers, as no backtracking is need. In contrast BM scales exponentially on this problem. For n-Queens, see Figure 4B, BM scale exponentially in n^2, while ZChaff seems to have better scalability. Again, a different picture emerges when we consider the pigeon-hole problem and the mutilated-checkerboard problem, see Figure 5A and Figure 5B. On both problems both BM and ZChaff scale exponentially, but BM scales exponentially better than ZChaff.

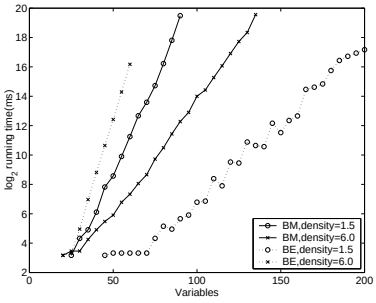
As in [55], who compared BDDs and DPLL for solution enumeration, we find that no approach dominates across all classes. While ZChaff dominates for many classes of formulas, the symbolic approach is superior for other classes of formulas. This suggests that the symbolic quantifier elimination is a viable approach and deserves further study. In the next section of this work we focus on various optimization strategies for the symbolic approach.

5. Optimizations

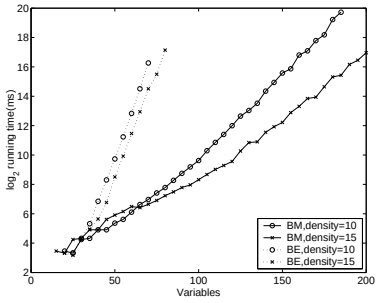
So far we have described one approach to symbolic quantifier elimination. There are, however, many choices one needs to make to guide an implementation. The order of variables is both used to guide clustering and quantifier elimination, as well as to order the variables in the underlying BDDs. Both clustering and cluster processing can be performed in several ways. In this section, we investigate the impact of choices in clustering, variable order, and quantifier elimination in the implementation of symbolic algorithms. Our focus here is on measuring the impact of variable order on BDD-based SAT solving; thus, the running time for variable ordering, which is polynomial for all algorithms, is not counted in our figures.

5.1. Cluster Ordering

As argued earlier, the purpose of quantifier elimination is to reduce support-set size of intermediate BDDs. What is the best reduction one can hope for? This question has been studied in the context of constraint satisfaction. It turns out that the optimal schedule of conjunctions and quantifier eliminations reduces the support-set size to one plus the *treewidth* of the Gaifman graph of the input formula [21]. The treewidth of a graph is a measure of how close this graph is to being a tree [28]. Computing the treewidth of a graph is known to be NP-hard, which is why heuristic approaches are employed [41]. It turns out that by processing clusters in a different order we can attain the optimal support-set size. Recall that BM processes the clusters in order of increasing ranks. *Bucket elimination* (BE), on the other hand, processes clusters in order of decreasing ranks [26]. Maximal support-size set of BE with respect to optimal variable order is defined as the *induced width* of the input instance, and the induced width is known to be equal to the treewidth [26, 30]. Thus, BE with respect to optimal variable order is guaranteed to have polynomial running time for input instances of logarithmic treewidth, since this guarantees a polynomial upper bound



A.



B.

Figure 6. Clustering Algorithms - Random 3-CNF

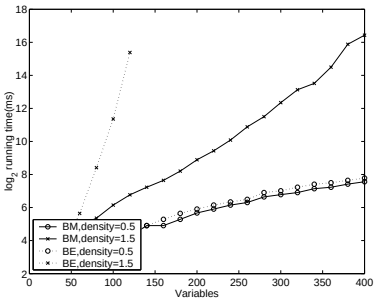


Figure 7. Clustering Algorithms - Random Affine

on BDD size. For BE, since the maximum-ranked variable in each cluster can not occur in any lower-ranked clusters, computing a quantification schedule from the contents of the clusters is not necessary. As each cluster is processed, the maximum-ranked variable is eliminated. For example, for the formula presented in Section 2, the quantification schedule would be $\exists x_1((\exists x_2((\exists x_3((\exists x_4 C_4) \wedge C_3)) \wedge C_2)) \wedge C_1)$, with one variable eliminated per cluster processed. As shown, using the inverse of variable rank as the BDD variable order allows us to always eliminate the top variable in the BDD.

We now compare BM and BE with respect to MCS variable order (MCS is the preferred variable order also for BE).

The results for the comparison on random 3-CNF formulas is plotted in Figure 6A and 6B. We see that the difference between BM and BE is density dependent, where BE excels in the low-density case, which have low treewidth, and BM excels in the high-density cases, which has high treewidth. A similar density dependent behavior is shown for the affine case in Figure 7. The difference of the two schemes on biconditional formulas is quite small as shown in Figure 8A. For chains, see Figure 8B. Because the number of variables for these formulas are large, the cost of computing the quantification schedule gives BE an edge over BM.

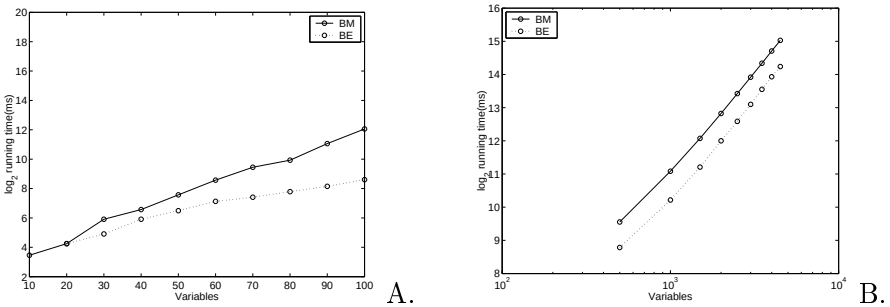


Figure 8. Clustering Algorithms - A) Random Biconditionals B) Random Chains

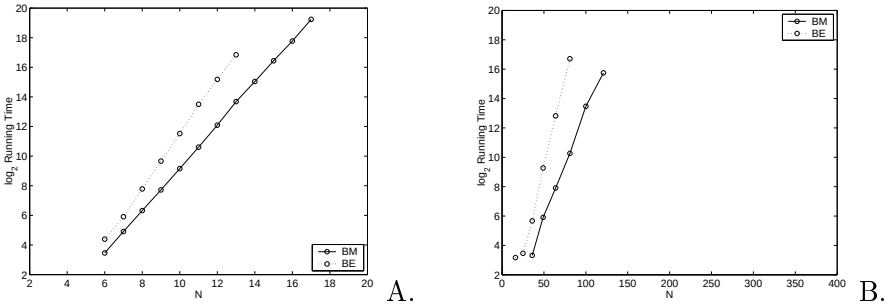


Figure 9. Clustering Algorithms - A) n-Rooks B) n-Queens

On most constructed formulas, the picture is similar to the high-density random cases, where BM dominates, except for mutilated-checkerboard formulas, where BE has a slight edge. (Note that treewidth for mutilated checkerboard problems grows only at $O(n)$ compared to $O(n^2)$ for other constructed problems.) We plot the performance comparison for n-rook formulas in Figure 9A, n-queens formulas in Figure 9B, pigeon-hole formulas in Figure 10A, and mutilated checkerboard problems in Figure 10B.

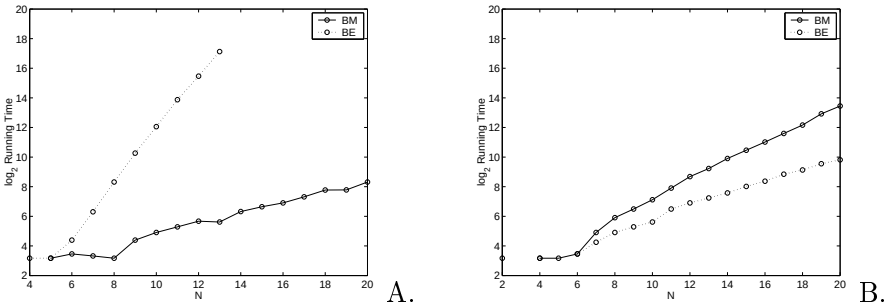


Figure 10. Clustering Algorithms - A) Pigeon Hole B) Mutilated Checkerboard

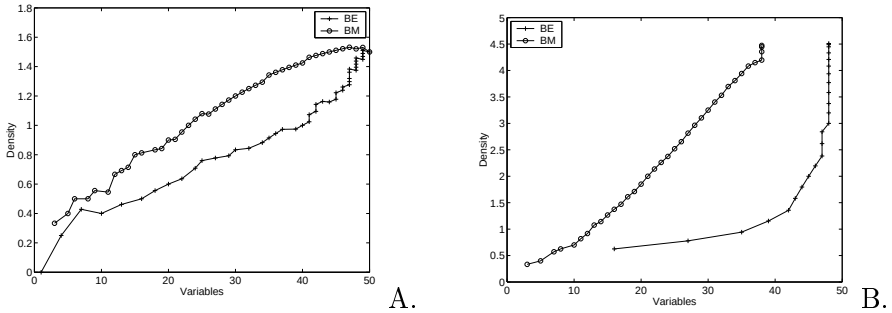


Figure 11. Clustering Algorithms A) Density=1.5 B) Density=6.0

To understand the difference in performance between BM and BE, we study their effect on intermediate BDD size. BDD size for a random 3-CNF instance depends crucially on both the number of variables and the density of the instance. Thus, we compare the effect of BM and BE in terms of these measures for the intermediate BDDs. We apply BM and BE to random 3-CNF formulas with 50 variables and densities 1.5 and 6.0. We then plot the density vs. the number of variables for the intermediate BDDs generated by the two cluster-processing schemes. The results are plotted in in Figure 11A and Figure 11B. Each plotted point corresponds to an intermediate BDD, which reflects the clusters processed so far.

As can be noted from the figures, BM increases the density of intermediate results much faster than BE. This difference is quite dramatic for high-density formulas. The relation between density of random 3-CNF instance and BDD size has been studied in [19], where it is shown that BDD size peaks at around density 2.0, and is lowest when the density is close to 0 or the satisfiability threshold. This enables us to offer an possible explanation to the superiority of BE for low-density instances and the superiority of BM for high-density instances. For formulas of density 1.5, the density of intermediate results is smaller than 2.0 and BM's increased density results in larger BDDs. For formulas of density 6.0, BM crosses the threshold density 2.0 using a smaller number of variables, and then BM's increased density results in smaller BDDs.

The general superiority of BM over BE suggests that minimizing support-set size ought not to be the dominant concern. BDD size is correlated with, but not dependent on, support-set size. More work is required in order to understand the good performance of BM. Our explanation argues that, as in [3], BM first deals with the most constrained subproblems, therefore reducing BDD-size of intermediate results. While the performance of BE can be understood in terms of

treewidth, we still lack, however, a fundamental theory to explain the performance of BM.

5.2. Variable Ordering

In this section, we study the effects of the variable order on the performance of symbolic algorithms. We only present results for BM, since the picture is similar for BE. The variable order for the BDD representation is again the inverse of the variable order for clustering. As mentioned earlier, when selecting variables, MCS has to break ties, which happens quite often. One can break ties by choosing (from those variables that have the maximum cardinality to ranked variables as MCS requires) the variable with minimal degree to unselected variables [50] or the variable with the maximal degree to unselected variables [6]. (Another choice to break ties uniformly at random, but this choice is expensive to implement, since it is difficult to choose an element uniformly at random from a heap.) We compare these two heuristics with an arbitrary tie-breaking heuristic, in which we simply select the top variable in the heap. The results are shown in Figure 12A for random 3-CNF formulas. For high density formulas, tie breaking made no significant difference, but least-degree tie breaking is markedly better for the low density formulas. This seems to be applicable across a variety of class of formulas and even for different orders and algorithms.

MCS typically has many choices for the lowest-rank variable. In Koster et. al. [41], it is recommended to start from every vertex in the graph and choose the variable order that leads to the lowest treewidth. This is easily done for instances of small size, i.e. random 3-CNF or affine problems; but for structured problems, which could be much larger, the overhead is too expensive. Since min-degree tie-breaking worked quite well, we used the same idea for initial variable choice. In Figure 12B, we see that our assumption is well-founded, that is, the benefit of choosing the best initial variable compared to choosing a min-degree variable is negligible. For larger problems like the chains or the bigger constructed problems, the additional overhead of trying every initial variable would be prohibitive, so we used the low-degree seed in all cases.

Algorithms for BDD variable ordering in the model-checking systems are often based on circuit structures, for example, some form of circuit traversal [31, 43] or graph evaluation [16]. These techniques are not applicable here, since the formulas are provided in CNF and the original circuit structure is lost.

MCS is just one possible vertex-ordering heuristics. Other heuristics have been studied in the context of treewidth approximation. In

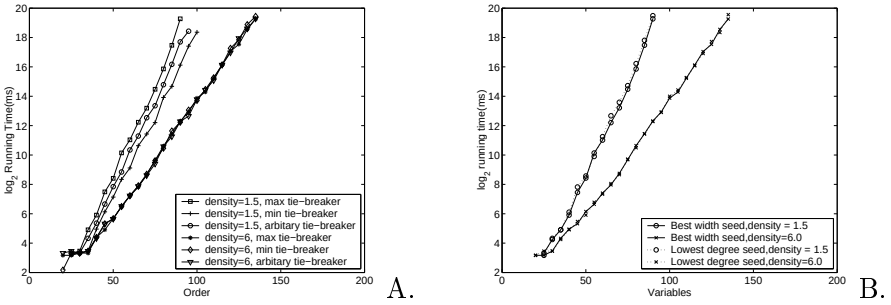


Figure 12. A) Variable Ordering Tie-breakers B) Initial Variable Choice

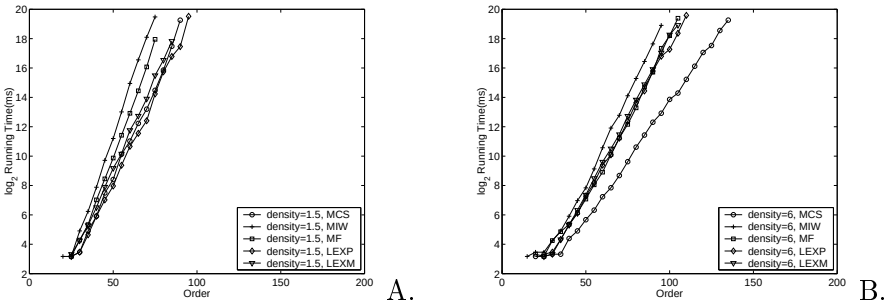


Figure 13. Vertex Order Heuristics - Random 3-CNF A) Density=1.5 B) Density=6

[41] two other vertex-ordering heuristics that based on local search are studied: LEXP and LEXM. Both LEXP and LEXM are based on *lexicographic breadth-first search*, where candidate variables are lexicographically ordered with a set of labels, and the labels are either the set of already chosen neighbors (LEXP), or the set of already chosen vertices reachable through lower-ordered vertices (LEXM). Both algorithms try to generate vertex orders where a triangulation would add a small amount of edges, thus reducing treewidth. In [25], Dechter also studied heuristics like Min-Induced-Width (MIW) or Min-Fill (MF), which are greedy heuristics based on choosing the vertex that have the least number of induced neighbors (MIW) or the vertex that would add the least number of induced edges (MF).

In Figure 13A and 13B, we compare variable orders constructed from MCS, LEXP, LEXM, MIW, and MF for random 3-CNF formulas. For high-density cases, MCS is clearly superior. For low-density formulas, LEXP has a small edge, although the difference is quite minimal. Across the other problem classes (for example, pigeon-hole formulas as in Figure 14A and mutilated checkerboard as in Figure 14B), MCS uniformly appears to be the best order, being the most consistent and generally the top performer. Interestingly, while other heuristics like MF often yield better treewidth, MCS still yields better runtime performance.

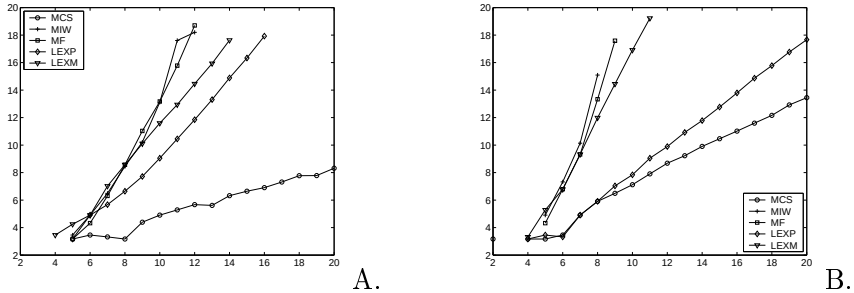


Figure 14. Vertex Order Heuristics - A) Pigeon Hole B) Mutilated Checkerboard

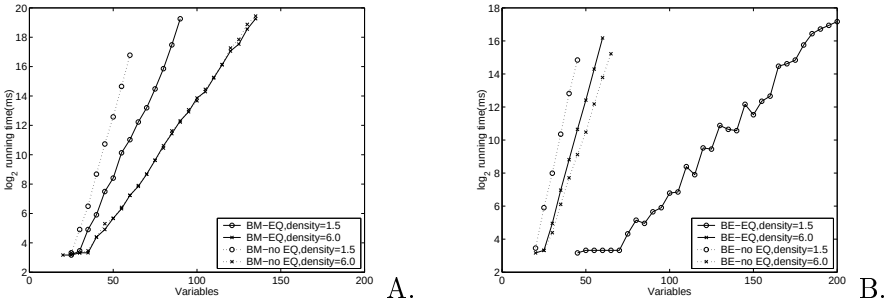


Figure 15. Quantifier Elimination-Random 3-CNF

This indicates that minimizing treewidth need not be the dominant concern; the dominant concern is minimizing BDD size. (BDD size seems more closely related to *pathwidth* [10], rather than treewidth. We speculate that MCS is a better order for pathwidth minimization.)

5.3. Quantifier Elimination

So far we argued that quantifier elimination is the key to the performance of the symbolic approach. In general, reducing support-set size does result in smaller BDDs. It is known, however, that quantifier elimination may incur non-negligible overhead and may not always reduce BDD size [12]. To understand the role of quantifier elimination in the symbolic approach, we reimplemented BM and BE without quantifier elimination. Thus, we do construct a BDD that represent all satisfying truth assignments, but we do that according to the clustering and cluster processing order of BM and BE.

In Figure 15A and 15B, we plotted the running time of both BM and BE, with and without quantifier elimination on random 3-CNF formulas. We see that there is a trade-off between the cost and benefit of quantifier elimination. For low-density instances, where there are many solutions, the improvement from quantifier elimination is clear, but for high-density instances, quantifier elimination results in no improvement

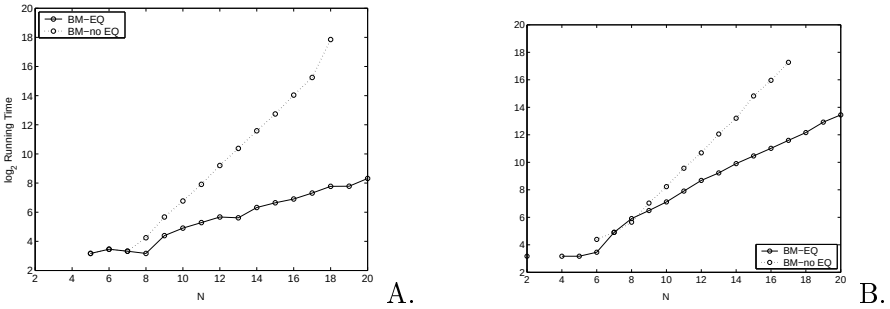


Figure 16. Quantifier Elimination - A) Pigeon Hole B) Mutilated Checkerboard

(while not reducing BDD size). For BE, where the overhead of quantifier elimination is lower, quantifier elimination improves performance very significantly at low density, although at high density there is a slight slow down. On the other hand, quantifier elimination is important for the constructed formulas, for example, for the pigeon-hole formulas in Figure 16A and the mutilated checkerboard formulas in Figure 16B.

6. Comparison with other approaches

In the previous section, we conducted a comprehensive comparison of the impact of different parameters on the BDD-based symbolic approach. Next, we expand our focus to alternate approaches, first by comparing the BDD-based symbolic quantifier elimination with ZDD-based multi-resolution, then compare the structural variable order we used with the default dynamic variable order in the context of ZChaff.

6.1. BDDs vs. ZDDs

So far we used symbolically represented sets of truth assignments. An alternate approach is to use decision diagrams to represent sets of clauses instead of sets of assignments. ZRes [15] is a symbolic implementation of the directional resolution algorithm in [24, 27]. The approach is also referred to as *multi-resolution*, since the algorithm carries out all resolutions over a variable in one symbolic step. Since individual clauses are usually sparse with respect to the set of variables, ZRes [15] used ZDDs [44], which typically offer a higher compression ratio then BDDs for the sparse spaces. Each propositional literal ℓ is represented by a ZDD variable v_ℓ (thus a propositional variable can be represented by two ZDD variables), and clause sets are represented as follows:

– The empty clause ϵ is represented by the terminal node 1.

- The empty set $\emptyset$ is represented by the terminal node 0.
- Given a set C of clauses and a literal ℓ whose ZDD variable v_ℓ is lowest in a given variable order, we split C into two subsets: $C_\ell = \{c \mid c \in C, \ell \in c\}$ and $C' = C - C_\ell$. Given ZDDs representing $C'' = \{c \mid c \vee \ell \in C_\ell\}$ and C', a ZDD representing C would be rooted at v_ℓ and have ZDDs for C'' and C' as its left and right children.

This representation is the dual of using ZDDs to represent Irredundant Sum of Products (ISOPs) of Boolean functions [44].

We use two set operations on sets of clauses: (1) $\times$ is the crossproduct operator, where for two clause sets C and D, $C \times D = \{c \mid \exists c' \in C, \exists c'' \in D, c = c' \cup c''\}$, and (2) $+$ is subsumption-free union, so if both C and D are subsumption free, and $c \in C + D$, then there is no $c' \in C + D$ where $c' \subset c$. Multi-resolution is implemented using $\times$ on cofactors: given a ZDD f, f_{x^+} (resp. f_{x^-}) is the ZDDs corresponding to the positive cofactor on the ZDD variable v_x (resp., $v_{\neg x}$, so $f_{x^+} = \{a \mid a \vee x \in f\}$ and $f_{x^-} = \{a \mid a \vee \neg x \in f\}$. Now $f_{x^+} \times f_{x^-}$ (after removing tautologies) represents the set of all resolvents of f on x, which has to be combined using $+$ with $f_{x'}$, which is the ZDD for the clauses not containing x. ZRes eliminates variables using multi-resolution one by one until either the empty clause is generated, in which case the formula is unsatisfiable, or all variables have been eliminated, in which case the formula is satisfiable.

To facilitate a fair comparison between ZRes and our BDD-based solver, we used the multi-resolution code used in [15] under our bucket-elimination framework and used the same variable and elimination order as the BDD-based algorithms. This can be seen as a comparison of the compression capability of ZDD-based clause sets versus BDD-based solutions sets representations, since at comparable stages of the two algorithms (say, before variable x_i is eliminated), the data structures represents the same Boolean function. As an optimization, a simple form of unit preference is implemented for the ZDD-based multi-resolution, since unit clauses can be easily detected in the ZDD-based clause set representation and resolved out-of-order.

The results for the 3-CNF and affine satisfiability cases are plotted in Figures 17A, 17B, and 18. We see that the differences between the two approaches are again density dependent. Just like the differences between BE and BM, ZDD-based multi-resolution is more efficient at low density and less efficient at high density. This can be related to the compression ratio achieved by the two representations at different densities, where the clause set representation is far more efficient at low densities. For the high-density case, the clause set representation starts

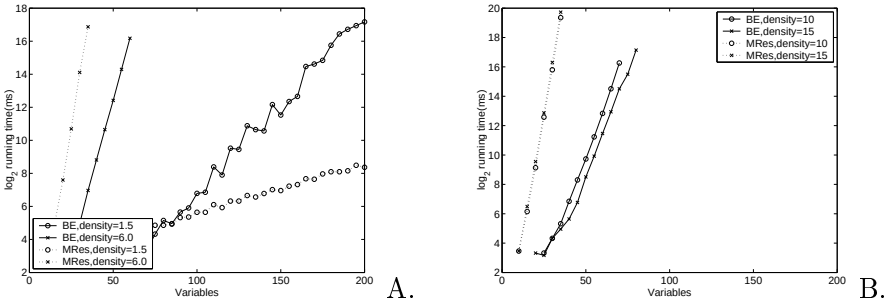


Figure 17. Random 3-CNF

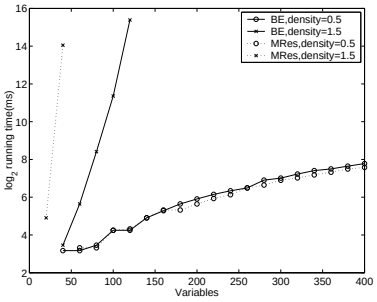


Figure 18. Random 3-Affine

to show its shortcomings. High-density problems typically have a large number of clauses and few solutions, clause-set representation is less efficient in this case. This is especially evident for the unsatisfiable case, where if BDDs are used, unsatisfiability can be detected immediately, but if clause sets are used, detection is delayed until an empty clause is generated.

Next we examine the other classes of formulas in Figure 19A, 19B, 20A, 20B, 21A, and 21B. In all cases, the BDD-based approach is superior to the ZDD-based approach.[6]

An explanation for the superiority of the BDD-based approach can be provided in terms of the cost of the quantifier-elimination operation. Complexity of decision-diagram algorithms can be measured in the number of cache look-ups that the algorithm performs. Quantifying out a single variable uses the BDD "or" operation, which has a proven $O(n^2)$ upper bound on the number of cache look-ups [12]. The same cannot be said for the ZDD multi-resolution operation used to quantify out a single variable, where the number of cache look-ups can be

[6] There exists other ZDD-based approaches for hard-for-resolution problems, for example, CASSAT [46], which exhibits polynomial running time on pigeon-hole formulas [47]. A comparison against these approaches would be a future direction of this research.

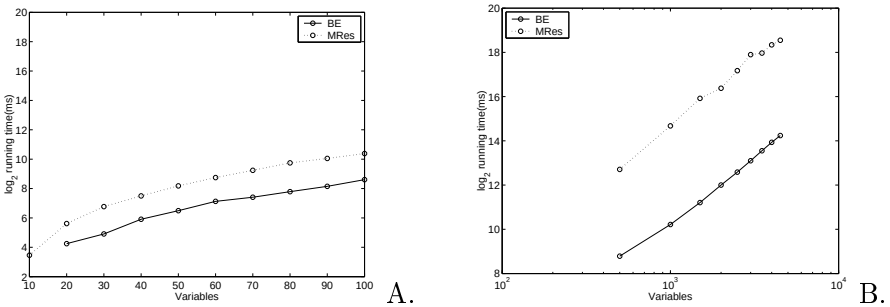


Figure 19. A) Random Biconditionals B) Random Chains

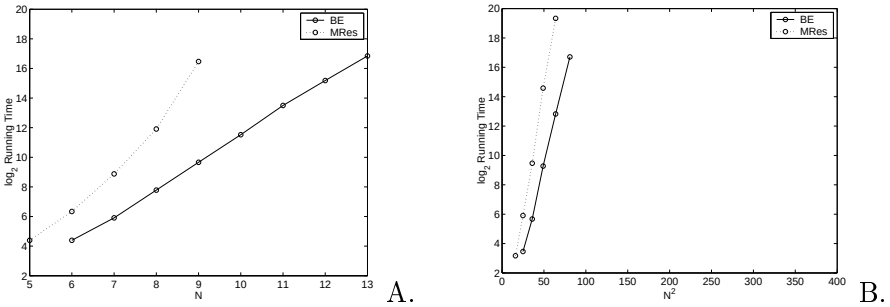


Figure 20. A) n-Rooks B) n-Queens

exponential in the width of the input ZDDs. Empirically, the number of cache lookups can be 1-2 orders of magnitude larger than the size of the output ZDD. This is the main contribution to the performance hit taken by the ZDD-based algorithm.

In [48] we compared BDD-based and ZDD-based approaches to QBF solving, showing that ZDD-based multi-resolution has a clear edge. Since QBF problems are required to be under-constrained propositionally (otherwise they would be easily unsatisfiable due to the universal quantifiers), the extra compression of the ZDD-based clause-set rep-

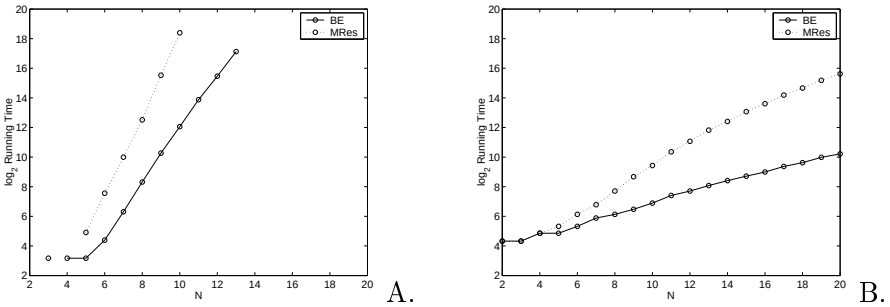


Figure 21. A) Pigeon Hole B) Mutilated Checkerboard

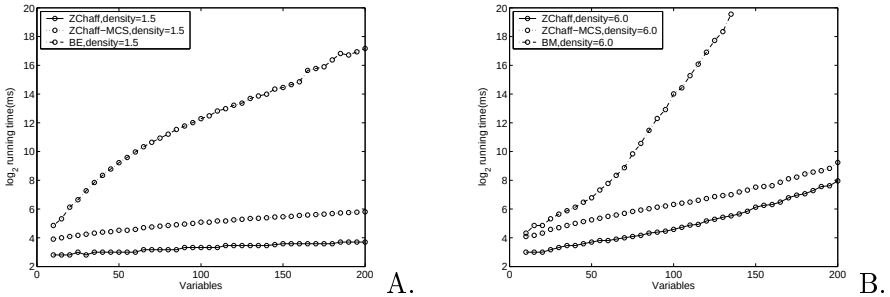


Figure 22. Variable Order - Random 3-CNF (1)

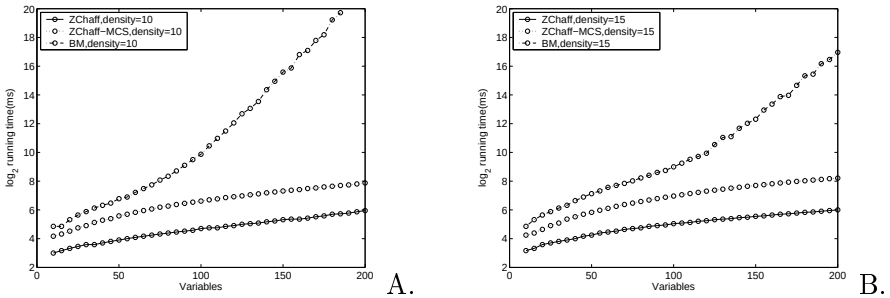


Figure 23. Variable Order - Random 3-CNF (2)

resentation would apply, explaining the superiority of the ZDD-based approach.

6.2. Structure-Guided Variable Order for Search

In Section 5, we showed that the choice of variable order is important to the performance of BDD-based satisfiability solvers. We showed that MCS variable order offers good algorithmic performance across a variety of input formulas. In contrast, most search-based algorithm use a dynamic variable order, based on the clauses visited or generated during the search procedure, for example, the VSIDS heuristic used in ZChaff [45]. To offer a more direct comparison between search-based and symbolic methods, we re-implemented ZChaff with the MCS variable order and compared its performance with ZChaff and with the symbolic solvers. (See [1, 37] for earlier work on structure-guided variable order for search-based methods.) We compared here the performance of ZChaff with the default (VSIDS) variable order, ZChaff with MCS variable order, and the BDD-based solvers (for each formula class we chose the best solver between BM and BE).

The results for random formulas are shown in Figures 22A, 22B, 23A, 23B, 24A, 24B, 25A, and 25B, and the results for constructed

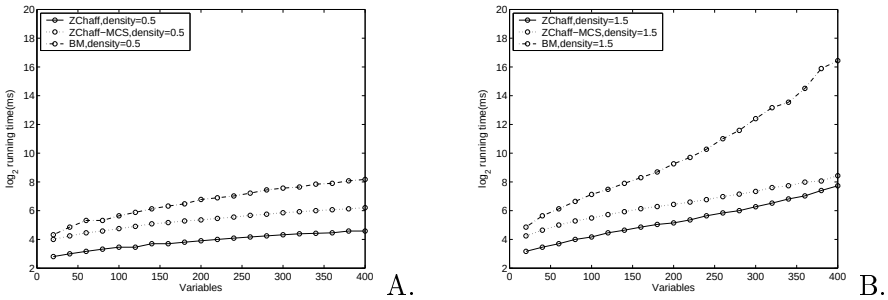


Figure 24. Variable Order - Random Affine

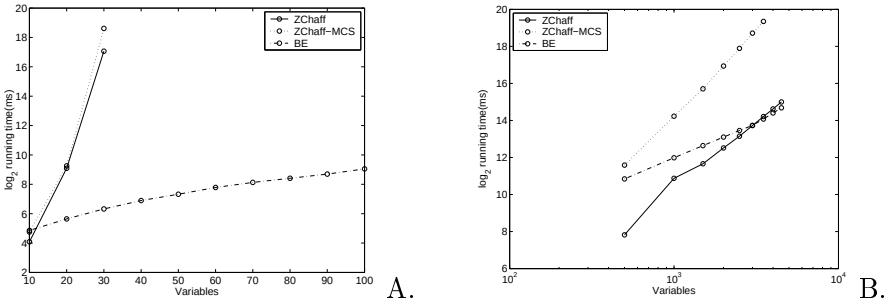


Figure 25. Variable Order - A) Random Biconditional B) Random Chains

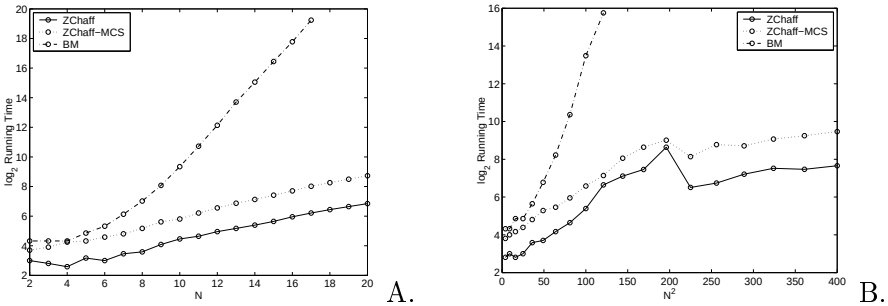


Figure 26. Variable Order - A) n-Rooks B) n-Queens

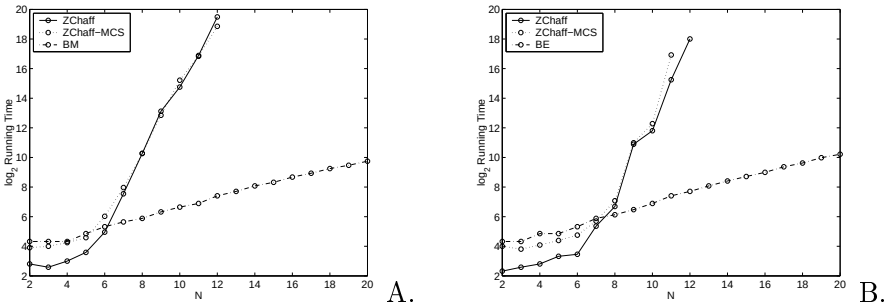


Figure 27. Variable Order - A) Pigeon Hole B) Mutilated Checkerboard

formulas are shown in Figures 26A, 26B, 27A, and 27B. In general, the structure-guided variable order is inferior in terms of performance to dynamic variable order (VSIDS). For easy problems, the overhead of pre-computing the variable order is quite significant. The performance loss should not be entirely attributed to the overhead though, since we also observed an increase in the number of implications performed. Thus, dynamic variable order is, in general, a better algorithmic choice. Nevertheless, for most formulas, there is no exponential gap in scaling between the two variable-order heuristics.

Also, replacing VSIDS by MCS did not change the relationship between ZChaff and the BDD-based solvers. The difference in performance between search-based and symbolic approaches is larger than the difference between static and dynamic decision order for ZChaff. In none of the cases did the static variable order change the relative picture between search and symbolic approaches. This shows the general superiority of search-based vs. symbolic techniques cannot be attributed to the use of dynamic variable order.

7. Discussion

Satisfiability solvers have made tremendous progress over the last few years, partly driven by frequent competitions, cf. [42]. At the same time, our understanding of why extant solvers perform so well is lagging. Our goal in this paper is not to present a new competitive solver, but rather to call for a broader research agenda in satisfiability solving. We showed that a symbolic approach can outperform a search-based approach in certain cases, but more research is needed before we can have robust implementations of the symbolic approach. Recent works have suggested other symbolic approaches to satisfiability solving, e.g., compressed BFS search in [46] and BDD representation for non-CNF constraint in the framework of DPLL search in [22, 29, 38]. These works bolster our call for a broader research agenda. Such an agenda should build connections with two other successful areas of automated reasoning, namely model checking [18] and constraint satisfaction [25]. Furthermore, such an agenda should explore *hybrid* approaches, combining search and symbolic techniques, cf. [22, 29, 35, 38, 46]. One hybrid approach that has shown promise is that of the QBF solver Quantor [7], where quantifier elimination is applied until the formula become propositional, then a search-based solver takes over.

As an extension to this work, we can experiment with other variable-order heuristics, for example, MINCE [1], FORCE [2], or the ones proposed in [37], all of which are also structurally based. Another direction

for development is to take a combination of density-dependent heuristics and structural heuristics and apply them to hybrid BDD-based SAT solvers like CirCUs [38] or the approach presented in [22].

Acknowledgements

We would like to thank Enrico Giunchiglia for proposing the experiments on structural-guided variable order for search.

References

1. Aloul, F., I. Markov, and K. Sakallah: 2001, 'MINCE: A Static Global Variable-Ordering for SAT and BDD'. In: *Proc. IEEE 10th International Workshop on Logic and Synthesis*. pp. 281–286.
2. Aloul, F., I. Markov, and K. Sakallah: 2003, 'FORCE: a fast and easy-to-implement variable-ordering heuristic'. In: *Proc. of the 13th ACM Great Lakes Symposium on VLSI 2003*. pp. 116–119.
3. Amir, E. and S. McIlraith: June 2001, 'Solving Satisfiability using Decomposition and the Most Constrained Subproblem'. In: *LICS Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*.
4. Arnborg, S., D. Corneil, and A. Proskurowski: 1987, 'Complexity of finding embeddings in a k-tree'. *SIAM J. Alg. Disc. Math* **8**, 277–284.
5. Balcazar, J.: 1990, 'Self-reducibility'. *J. Comp. and Sys. Sci.* **41**(3), 367–388.
6. Beatty, D. and R. Bryant: 1994, 'Formally verifying a microprocessor using a simulation methodology'. In: *Proc. 31st Design Automation Conference*. pp. 596–602.
7. Biere, A.: 2004, 'Resolve and Expand'. In: *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*. pp. 238–246.
8. Biere, A., C. A, E. Clarke, M. Fujita, and Y. Zhu: 1999, 'Symbolic Model Checking using SAT Procedures instead of BDD'. In: *Proc. 36th Conf. on Design Automation*. pp. 317–320.
9. Block, M., C. Gröpl, H. Preuß, H. L. Proömel, and A. Srivastav: 1997, 'Efficient ordering of state variables and transition relation partitions in symbolic model checking'. Technical report, Institute of Informatics, Humboldt University of Berlin.
10. Bodlaender, H. and T. Kloks: 1996, 'Efficient and constructive algorithms for the pathwidth and treewidth of graphs'. *J. Algorithms* **21**, 358–402.
11. Bouquet, F.: 1999, 'Gestion de la dynamicite et enumeration d'implicants premiers, une approche fondee sur les Diagrammes de Decision Binaire'. Ph.D. thesis, Universite de Privence, France.
12. Bryant, R.: 1986, 'Graph-based Algorithms for Boolean Function Manipulation'. *IEEE Trans. on Comp.* **Vol. C-35**(8), 677–691.
13. Burch, J., E. Clarke, and D. Long: 1991, 'Symbolic model checking with partitioned transition relations'. In: *VLSI 91, Proc. IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration, Edinburgh, Scotland, 20-22 August, 1991*. pp. 49–58.

14. Burch, J., E. Clarke, K. McMillan, D. Dill, and L. Hwang: 1992, 'Symbolic Model Checking: 10^{20} States and Beyond'. *Information and Computation* **98**(2), 142–170.
15. Chatalic, P. and L. Simon: 2000, 'Multi-Resolution on Compressed Sets of Clauses'. In: *Twelfth International Conference on Tools with Artificial Intelligence (ICTAI'00)*. pp. 2–10.
16. Chung, P., I. Hajj, and J. Patel: 1993, 'Efficient Variable Ordering Heuristics for Shared ROBDD'. In: *Proc. 1993 IEEE Int. Symp. on Circuits and Systems (ISCAS93)*. pp. 1690–1693.
17. Cimatti, A. and M. Roveri: 2000, 'Conformant Planning via Symbolic Model Checking'. *J. of AI Research* **13**, 305–338.
18. Clarke, E., O. Grumberg, and D. Peled: 1999, *Model Checking*. MIT Press.
19. Coarfa, C., D. D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M. Vardi: 2003, 'Random 3-SAT: The Plot Thickens'. *Constraints* pp. 243–261.
20. Crawford, J. and A. Baker: 1994, 'Experimental results on the application of satisfiability algorithms to scheduling problems'. In: *Proc. 12th Nat. Conf. on Artificial Intelligence*, Vol. 2. pp. 1092–1097.
21. Dalmau, V., P. Kolaitis, and M. Vardi: 2002, 'Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics'. In: *Proceedings of 8th Int. Conf. on Principles and Practice of Constraint Programming (CP 2002)*. pp. 310–326.
22. Damiano, R. F. and J. H. Kukula: 2003, 'Checking satisfiability of a conjunction of BDDs'. In: *Proc. 40th Design Automation Conference (DAC 2003)*. pp. 818–823.
23. Davis, M., G. Logemann, and D. Loveland: 1962, 'A machine program for theorem proving'. *J. ACM* **5**, 394–397.
24. Davis, S. and M. Putnam: 1960, 'A computing procedure for quantification theory'. *J. ACM* **7**, 201–215.
25. Dechter, R.: 2003, *Constraint Processing*. Morgan Kaufmann.
26. Dechter, R. and J. Pearl: 1987, 'Network-based heuristics for constraint-satisfaction problems'. *Artificial Intelligence* **34**, 1–38.
27. Dechter, R. and I. Rish: 1994, 'Directional Resolution: The Davis-Putnam Procedure, Revisited'. In: *KR'94: Principles of Knowledge Representation and Reasoning*. pp. 134–145.
28. Downey, R. and M. Fellows: 1999, *Parametrized Complexity*. Springer-Verlag.
29. Franco, J., M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W. Vanfleet: 2003, 'SBSAT: a State-based, BDD-based satisfiability solver'. In: *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003)*. pp. 398–410.
30. Freuder, E.: 1990, 'Complexity of k-tree structured constraint satisfaction problems'. In: *Proc. 8th Nat. Conf. on Artificial Intelligence*. pp. 4–9.
31. Fujita, M., H. Fujisawa, and N. Kawato: 1988, 'Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Disgrams'. In: *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD-88)*. pp. 2–5.
32. Geist, D. and H. Beer: 1994, 'Efficient Model Checking by Automated Ordering of Transition Relation Partitions'. In: *Proc. 6th Int. Conf. on Computer Aided Verification (CAV 1994)*. pp. 299–310.
33. Goldberg, E. and Y. Novikov: 2002, 'BerkMin: A Fast and Robust SAT Solver'. In: *Proc. Design Automation and Test in Europe (DATE 2002)*. pp. 142–149.
34. Groote, J. F.: 1996, 'Hiding Propositional Constants in BDDs'. *FMSD* **8**, 91–96.

35. Gupta, A., Z. Yang, P. Ashar, L. Zhang, and S. Malik: 2001, 'Partition-based decision heuristics for image computation using SAT and BDDs'. In: *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD-01)*. pp. 286–292.
36. Hojati, R., S. C. Krishnan, and R. K. Brayton: 1996, 'Early Quantification and Partitioned Transition Relations'. In: *Proc. 1996 Int. Conf. on Computer Design (ICCD '96)*. pp. 12–19.
37. Huang, J. and A. Darwiche: 2003, 'A Structure-Based Variable Ordering Heuristic for SAT'. In: *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*. pp. 1167–1172.
38. Jon, H. and F. Somenzi: 2004, 'CirCUs : Hybrid Satifiability solver'. In: *Proc. of the 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*. pp. 47–55.
39. Kautz, H. and B. Selman: 1992, 'Planning as satisfiability'. In: *Proc. 10th Eur. Conf. on AI (ECAI 92)*. pp. 359–363.
40. Khurshid, S., D. Marinov, I. Shlyyakhter, and D. Jackson: 2003, 'A Case for Efficient Solution Enumeration'. In: *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003)*. pp. 272–286.
41. Koster, A., H. Bodlaender, and S. van Hoesel: 2001, 'Treewidth: Computational Experiments'. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin.
42. Le Berre, D. and L. Simon: 2003, 'The essentials of the SAT'03 Competition'. In: *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003)*. pp. 452–467.
43. Malik, S., A. Wang, R. Brayton, and A. Sangiovanni Vincentelli: 1988, 'Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment'. In: *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD-88)*. pp. 6–9.
44. Minato, S.: 1996, *Binary Decision Diagrams and Applications to VLSI CAD*. Kluwer.
45. Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang, and S. Malik: 2001, 'Chaff: Engineering an Efficient SAT Solver'. In: *Proc. of 39th Design Automation Conference (DAC 2001)*. pp. 530–535.
46. Motter, D. B. and I. L. Markov: 2002a, 'A Compressed Breadth-First Search for Satisfiability'. In: *Proc. 4th Int. Workshop on Algorithm Engineering and Experiments (ALENEX 2002)*, Vol. 2409 of *Lecture Notes in Computer Science*. pp. 29–42.
47. Motter, D. B. and I. L. Markov: 2002b, 'On Proof Systems Behind Efficient SAT Solvers'. In: *Proc. of 5th Int. Symp. on the Theory and Applications of Satisfiability Testing (SAT 2002)*. pp. 206–213.
48. Pan, G. and M. Y. Vardi: 2004, 'Symbolic Decision Procedures for QBF'. In: *Proceedings of 10th Int. Conf. on Principles and Practice of Constraint Programming (CP 2004)*. pp. 453–467.
49. Ranjan, R., A. Aziz, R. Brayton, B. Plessier, and C. Pixley: 1995, 'Efficient BDD algorithms for FSM synthesis and verification'. In: *Proc. of IEEE/ACM Int. Workshop on Logic Synthesis*.
50. San Miguel Aguirre, A. and M. Y. Vardi: 2001, 'Random 3-SAT and BDDs: The Plot Thickens Further'. In: *Proc. of the 7th Int. Conf. Principles and Practice of Constraint Programming (CP 2001)*. pp. 121–136.
51. Schaefer, T.: 1978, 'The Complexity of Satisfiability Problems'. In: *Proc. of the 10th annual ACM symposium on Theory of computing(STOC'78)*. pp. 216–226.

52. Selman, B., D. G. Mitchell, and H. J. Levesque: 1996, 'Generating Hard Satisfiability Problems'. *Artificial Intelligence* **81**(1-2), 17–29.
53. Somenzi, F.: 1998, 'CUDD: CU Decision Diagram package'. http://vlsi.colorado.edu/~fabio/CUDD/.
54. Tarjan, R. E. and M. Yannakakis: 1984, 'Simple linear-time algorithms to tests chordality of graphs, tests acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs'. *SIAM Journal on Computing* **13**(3), 566–579.
55. Uribe, T. E. and M. E. Stickel: 1994, 'Ordered Binary Decision Diagrams and the Davis-Putnam Procedure'. In: *1st Int. Conf. on Constraints in Computational Logics*. pp. 34–49.
56. Urquhart, A.: 1995, 'The Complexity of Propositional Proofs'. *the Bulletin of Symbolic Logic* **1**, 425–467.