Search vs. Symbolic Techniques in Satisfiability Solving

Guoqiang Pan*, Moshe Y. Vardi*

Department of Computer Science, Rice University, Houston, TX gqpan,vardi@cs.rice.edu

Abstract. Recent work has shown how to use OBDDs 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.

1 Introduction

Propositional-satisfiability solving has been an active area of research through out the last 40 years, starting from the resolution-based algorithm in [19] and the search-based algorithm in [18]. 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 have been made in developing highly optimized DPLL solvers, leading to efficient solvers such as ZChaff [46] and BerkMin [28], 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 bounded model checking [5], planning [32], scheduling [16], 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 (OBDDs) [8], which are used successfully in model checking [10] and planning [13]. The zero-suppressed variant (ZDDs) is used in prime implicants enumeration [36]. A decision-diagram representation also enables easy satisfiability checking, which amounts to deciding whether it is different than the empty OBDD [8]. 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 [15]. Thus, the only published comparison between search and OBDD techniques [44] 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 OBDDs are superior.

Recent work has shown how to use OBDDs for satisfiability solving rather for enumeration [39]. 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 [38] and constraint satisfaction [21]. The focus in [39] 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 [39] show that OBDD-based algoithms behave quite differently than search-based algorithms, which makes them worthy of further investigation. (Other recent approaches reported using decision diagrams in satisfiability solving [11, 24, 37]. We discuss these works in our concluding remarks).

* 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.

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 [34], we focus here on *scalability*. That is, we focus on scalable classes of formulas and evaluate how performance scales with formula size. As in [44] 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 [39] 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 [39] and the second is the *bucket-elimination* (BE) approach of [21]. BE aims at reducing the size of the support set of the generated OBDDs 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 [23]. 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 OBDD order. Finding an optimal order is by itself a difficult problem (computing the treewidth of a given graph is NP-hard [2]), so one has to resort to various heuristics, cf. [33]. No heuristic seems to dominate across all classes of formulas, but the maximal-cardinality-search heuristic seems to offer the best overall performance.

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. We conclude with a discussion in Section 6.

2 Background

An 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 (OBDD) is a BDD with the constraint that the input variables are ordered and every path in the OBDD visits the variables in ascending order. We assume that all OBDDs are *reduced*, which means that where every node represents a distinct logic function. OBDDs constitute an efficient way to represent and manipulate Boolean functions [8], in particular, for a given variable order, OBDDs offer a canonical representation. Checking whether an OBDD is satisfiable is also easy; it requires checking that it differs from the predefined constant **0** (the empty OBDD). We used the CUDD package for managing OBDDs [42]. The *support set* of an OBDD is the set of variables labeling its internal nodes.

In [44, 15], OBDDs 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 an OBDD A_i for each clause c_i. (Since a clause excludes only one assignments to its variables, A_i is of linear size.) An OBDD 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 OBDDs A and B. Finally, the resulting OBDD B_m represents all satisfying assignments of the input formula.

We can apply existential quantification to an OBDD 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 OBDD 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

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

This means that after constructing the OBDD B_i, we can existentially quantify x_j before conjuncting B_i with $A_{i+1}, \ldots, A_m$.

This motivates the following change in the earlier OBDD-based satisfying-solving algorithm [39]: after constructing the OBDD 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 an OBDD 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 OBDD 1 or 0. Symbolic quantifier elimination was first applied to SAT solving in [29] (under the name of *hiding functions*) and tried on random 3-SAT instances. The work in [39] 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 OBDDs. This observation was first made in the context of symbolic model checking, cf. [9, 27, 31, 6]. 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 [38] 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 OBDDs of the clauses in the C to obtain an OBDD 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 [7] (the focus 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 is 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. Satisfiability solving using symbolic quantifier elimination combined with Bouquet's clustering is referred to in [39] as *Bouquet's Method*, which we abbreviate here is as BM.

We still have to chose a variable order. An order that is often used in constraint satisfaction [20] is the "maximum cardinality search" (MCS) order of [43], 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 number, select the vertex adjacent to the largest number of previously numbered vertices (ties can be broken in various ways). Our first experiment is a performance comparison of MCS-based BM to ZChaff.

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 [34], 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). For random formulas, experiments were performed using x86 emulation on the Rice Terascale Cluster[2], which is a large Linux cluster of Itanium II processors with 4GB of memory each. Non-random formula experiments used a Pentium-4 1.7Ghz desktop.

Our test suite includes the following classes of formulas:

[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. [3].

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

- 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 [41]).
- Random affine 3-CNF: Affine 3-CNF formulas 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 $l_1 \oplus l_2 \oplus l_3 = 1$. Each constraint is then converted into four clauses, 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 [45]. 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 [22]. 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 [5] and planning [32].) We use a similar generation parameters as in [22], 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 [44], 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, 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.

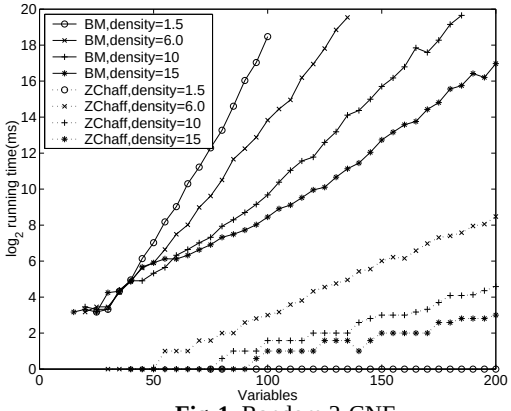

Fig. 1. Random 3-CNF

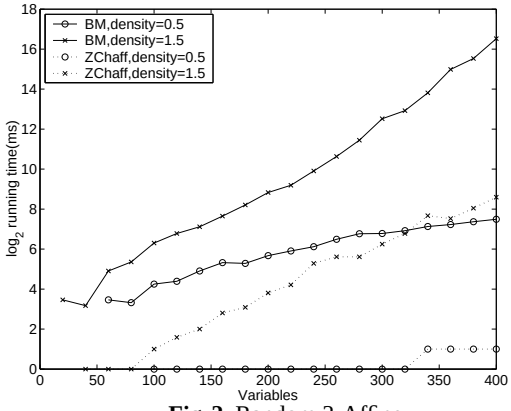

Fig. 2. Random 3-Affine

In Figure 1, 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 [15] and [39].) A similar pattern emerges for random affine formulas, 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 [40].)

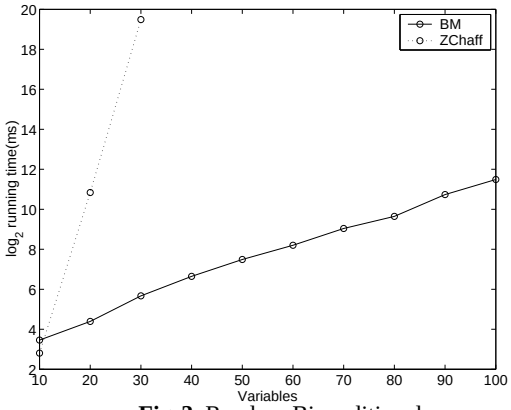

Fig. 3. Random Biconditionals

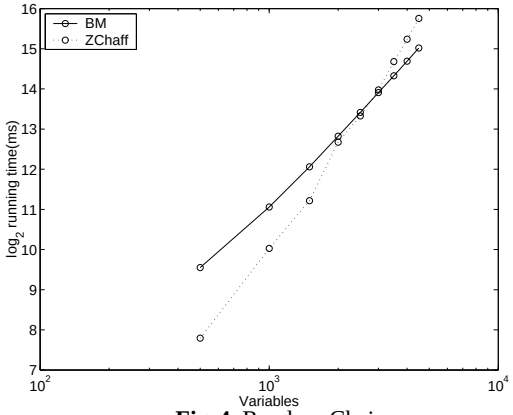

Fig. 4. Random Chains

The picture changes for biconditional formulas, as shown in Figure 3. Again, both methods are exponential, but BM scales exponentially better than ZChaff. (This result is consistent with the finding in [11], which compares search-based methods to ZDD-based multi-resolution.)

For random chains, see Figure 4, 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 than ZChaff. Note that for smaller instances ZChaff outperforms BM, which justifies our focus on scalability rather than on straightforward benchmarking.

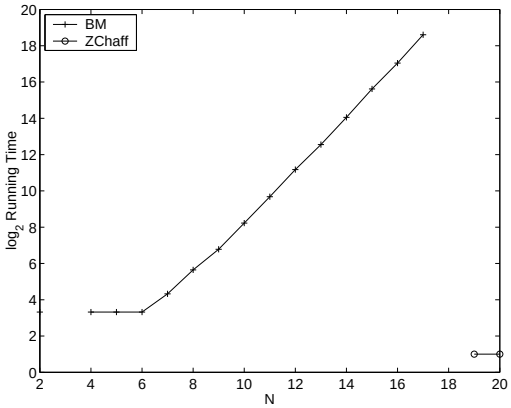

Fig. 5. n-Rooks

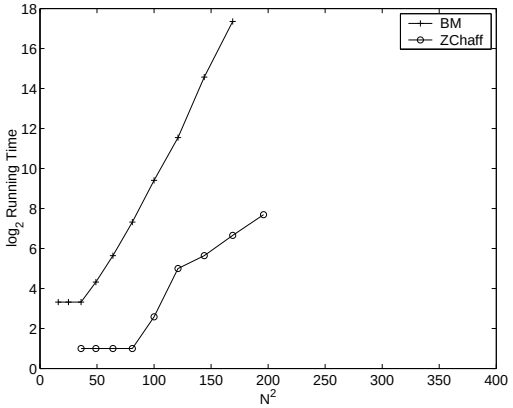

Fig. 6. n-Queens

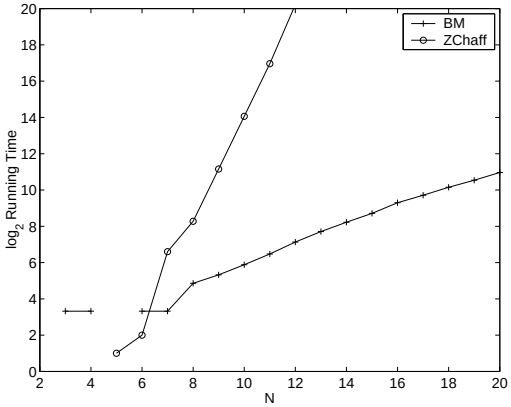

Fig. 7. Pigeon Hole

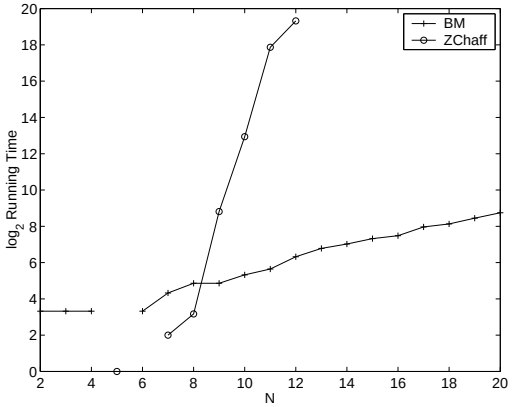

Fig. 8. Mutilated Checkerboard

Finally, we compare BM with ZChaff on the non-random formulas of [44]. 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 5. This problem have the property of being *globally consistent*, i.e., any consistent partial solution can be extended to a solution [20]. 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 6, both methods scale exponentially (in fact, they scale exponentially in n^2), but ZChaff scales exponentially

better than BM. Again, a different picture emerges when we consider the pigeon-hole problem and the mutilated-checkerboard problem, see Figure 7 and Figure 8. On both problems both BM and ZChaff scale exponentially, but BM scales exponentially better than ZChaff.

As in [44], who compared OBDDs 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 OBDDs. Both clustering and cluster processing can be performed in several ways. In this section, we investigate the impact of choices in clustering in the implementation of symbolic algorithms. For the impact of variable order and quantifier elimination, please refer to the appendix.

5.1 Cluster Ordering

As argued earlier, the purpose of quantifier elimination is to reduce support-set size of intermediate OBDDs. 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 [17]. The treewidth of a graph is a measure of how close this graph is to being a tree [23]. Computing the treewidth of a graph is known to be NP-hard, which is why heuristic approaches are employed [33]. 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 [21]. 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 [21, 25]. 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 on OBDD size. We now compare BM and BE with respect to MCS variable order (MCS is the preferred variable order also for BE).

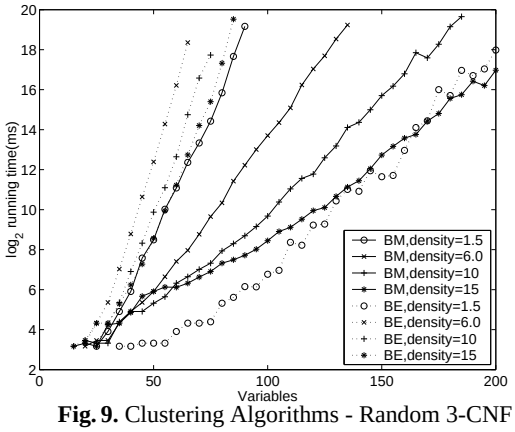

Fig. 9. Clustering Algorithms - Random 3-CNF

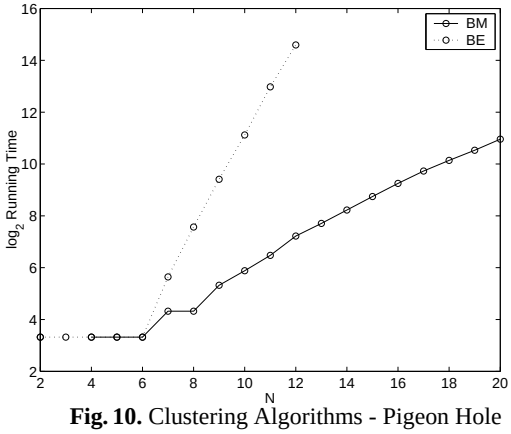

Fig. 10. Clustering Algorithms - Pigeon Hole

The results for the comparison on random 3-CNF formulas is plotted in Figure 9. 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. Across our other classes of random formulas, BM is typically better, except for a slight edge that BE sometimes has for low-density instances. A similar picture can be seen on most constructed formulas, where BM dominates, except for mutilated-checkerboard formulas, where BE has a slight edge. We plot the performance comparison for pigeon-hole formulas in Figure 10.

To understand the difference in performance between BM and BE, we study their effect on intermediate OBDD size. OBDD 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 OBDDs. 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 OBDDs generated by the two cluster-processing schemes. The results are plotted in in Figure 11 and Figure 12. Each plotted point corresponds to an intermediate OBDD, which reflects the clusters processed so far.

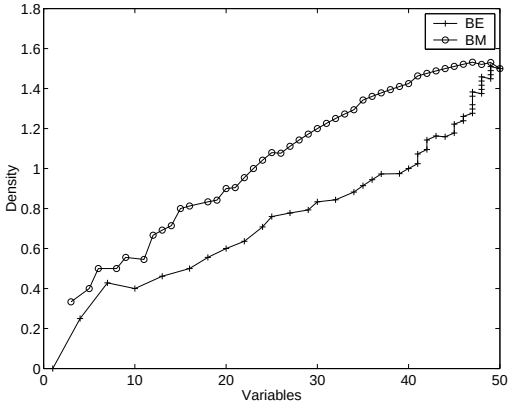

Fig. 11. Clustering Algorithms, Density=1.5

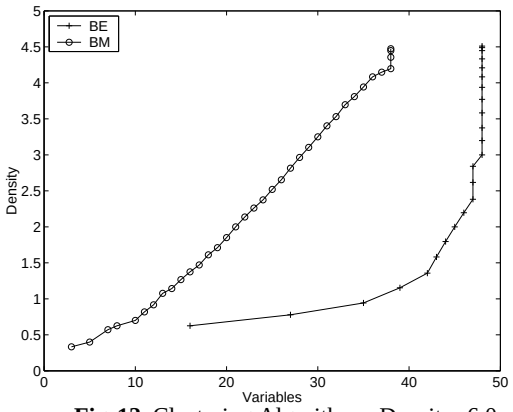

Fig. 12. Clustering Algorithms, Density=6.0

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 OBDD size has been studied in [15], where it is shown that OBDD 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 OBDDs. 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 OBDDs.

The general superiority of BM over BE suggests that minimizing support-set size ought not to be the dominant concern. OBDD 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 [1], BM first deals with the most constrained subproblems, therefore reducing OBDD-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

As mentioned earlier, when selecting variables, MCS has to break ties, which happens quite often. One can break ties by minimizing degree to unselected variables [39] or by maximizing it [4]. (Another choice to 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 heuristic with an arbitrary tie-breaking heuristic, in which we simply select the top variable in the heap. The results are shown in Figure 13 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. [33], 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 14, 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.

Algorithms for BDD variable ordering in the model checking area are often based on circuit structures, for example some form of traversal [35, 26] or graph evaluation [12]. Since we only have the graph structure

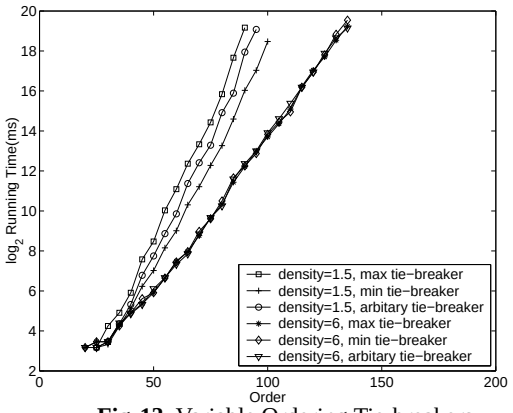

Fig. 13. Variable Ordering Tie-breakers

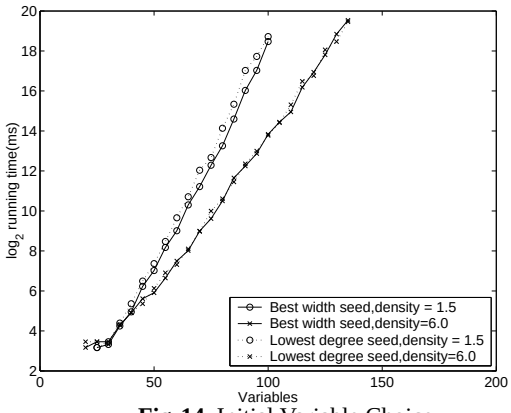

Fig. 14. Initial Variable Choice

based on the CNF clauses, we do not have the depth or direction information that circuit structure can provide. As the circuits in question become more complex, the effectiveness of simple traversals would also reduce. So, we use the graph-theoretic approaches used in constraint satisfaction instead of those from model checking.

MCS is just one possible vertex-ordering heuristics. Other heuristics have been studied in the context of treewidth approximation. In [33] two other vertex-ordering heuristics are studied: LEXP and LEXM.[3] Both LEXP and LEXM are based on *lexicographic breadth-first search*, where candidate variables are lexicographically ordered with a set of labels, where 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.

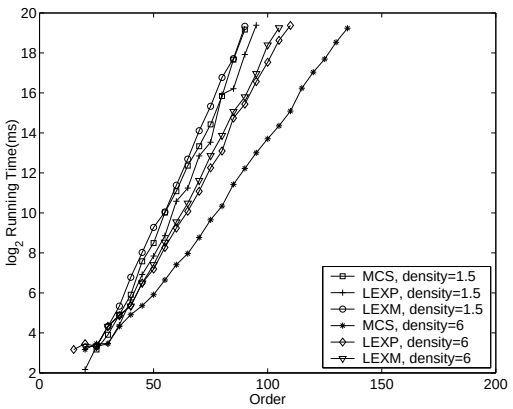

Fig. 15. MCS vs. LEX - Random 3-CNF

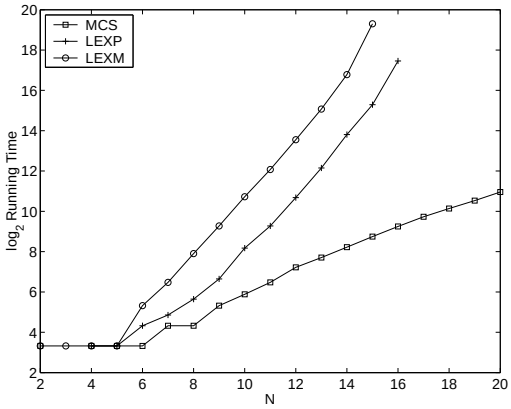

Fig. 16. MCS vs. LEX - Pigeon Hole

In Figure 15, we compare variable orders constructed from MCS, LEXP, and LEXM 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 16), MCS uniformly appears to be the best order, generally being the top performer. Interestingly, LEXP and LEXM sometimes yield better treewidth, but MCS till yields better runtime performance. This indicates that minimizing treewidth need not be the dominant concern.

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 OBDDs. It is known, however, that quantifier elimination may incur non-negligible overhead and may not always reduce OBDD size [8]. To understand the role of quantifier elimination in the symbolic approach, we reimplemented BM and BE without quantifier elimination. Thus, we do construct an OBDD that represent all satisfying truth assignments, but we do that according to the clustering and cluster processing order of BM and BE.

[3] The other heuristic mentioned in [33] is MSVS, which constructes a tree-decomposition instead of a variable order.

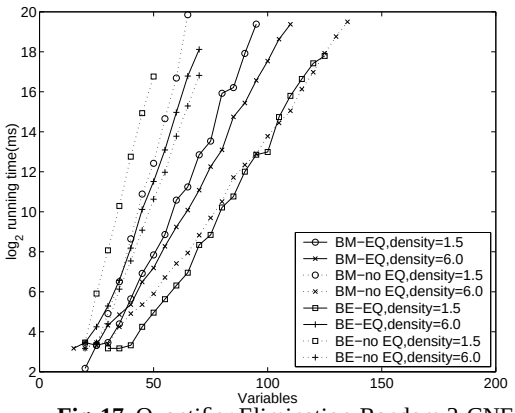

Fig. 17. Quantifier Elimination-Random 3-CNF

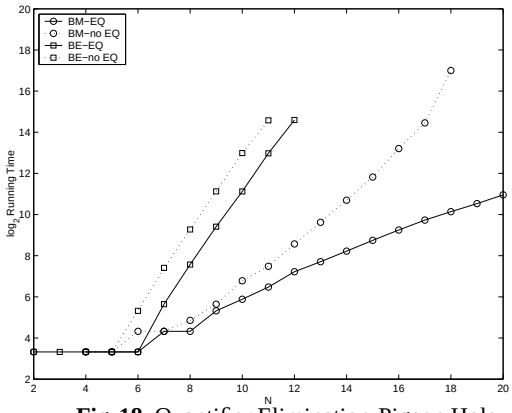

Fig. 18. Quantifier Elimination-Pigeon Hole

In Figure 17, we plotted the running time of both BM and BE, with and without quantifier elimination on random 3-CNF formulas. We see that for BM 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 slow down (while not reducing OBDD size). A similar picture holds for BE, though there the overhead of quantifier elimination is lower, making it a better choice. On the other hand, quantifier elimination is important for the constructed formulas, for example, for the pigeon-hole formulas in Figure 18.

6 Discussion

Satisfiability solvers have made tremendous progress over the last few years, partly driven by frequent competitions, cf. [34]. 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. We showed that a symbolic approach can outperform a search-based approach, but much 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., ZDD-based multi-resolution in [11], compressed BFS search in [37], and BDD representation for non-CNF constraint in the framework of DPLL search in [24]. These works bolster our call for a broader research agenda in satisfiability solving. Such an agenda should build connections with two other successful areas of automated reasoning, namely model checking [14] and constraint satisfaction [20]. Furthermore, such an agenda should explore *hybrid* approaches, combining search and symbolic techniques, cf. [30, 37, 24].

References

1. E. Amir and S. McIlraith. Solving satisfiability using decomposition and the most constrained subproblem. In *SAT 2001*, June 2001.
2. S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Math*, 8:277–284, 1987.
3. J. Balcazar. Self-reducibility. *J. Comput. Syst. Sci.*, 41(3):367–388, 1990.
4. D. Beatty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *Proc. 31st Design Automation Conference*, pages 596–602. IEEE Computer Society, 1994.
5. A. Biere, C. A, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDD. In *Proc. 36th Conf. on Design Automation*, pages 317–320, 1999.
6. M. Block, C. Gröpl, H. Preuß, H. L. Proömel, and A. Srivastav. Efficient ordering of state variables and transition relation partitions in symbolic model checking. Technical report, Institute of Informatics, Humboldt University of Berlin, 1997.
7. F. Bouquet. *Gestion de la dynamicite et enumeration d'implicants premiers, une approche fondee sur les Diagrammes de Decision Binaire*. PhD thesis, 1999.
8. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, Vol. C-35(8):677–691, August 1986.
9. J. Burch, E. Clarke, and D. Long. Symbolic model checking with partitioned transition relations. In *Int. Conf. on Very Large Scale Integration*, 1991.
10. J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Infomation and Computation*, 98(2):142–170, 1992.

11. P. Chatalic and L. Simon. Multi-Resolution on compressed sets of clauses. In *Twelfth International Conference on Tools with Artificial Intelligence (ICTAI'00)*, pages 2–10, 2000.
12. P. Chung, I. Hajj, and J. Patel. Efficient variable ordering heuristics for shared robdd. In *Proc. Int. Symp. on Circuits and Systems*, 1993.
13. A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *J. of AI Research*, 13:305–338, 2000.
14. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
15. C. Coarfa, D. D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M. Vardi. Random 3-SAT: The plot thickens. *Constraints*, pages 243–261, 2003.
16. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI*, volume 2, pages 1092–1097, 1994.
17. V. Dalmau, P. Kolaitis, and M. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP'02*, pages 310–326, 2002.
18. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.
19. S. Davis and M. Putnam. A computing procedure for quantification theory. *Journal of ACM*, 7:201–215, 1960.
20. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
21. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
22. R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *KR'94: Principles of Knowledge Representation and Reasoning*, pages 134–145. 1994.
23. R. Downey and M. Fellows. *Parametrized Complexity*. Springer-Verlag, 1999.
24. J. Franco, M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W. Vanfleet. SBSAT: a state-based, BDD-based satisfiability solver. In *SAT 2003*, 2003.
25. E. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. AAAI-90*, pages 4–9, 1990.
26. M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision disgrams. In *ICCAD*, 1988.
27. D. Geist and H. Beer. Efficient model checking by automated ordering of transition relation partitions. In *CAV 1994*, pages 299–310, 1994.
28. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver, 2002.
29. J. F. Groote. Hiding propositional constants in BDDs. *FMSD*, 8:91–96, 1996.
30. A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik. Partition-based decision heuristics for image computation using SAT and BDDs. In *ICCAD*, 2001.
31. R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. pages 12–19, 1996.
32. H. Kautz and B. Selman. Planning as satisfiability. In *Proc. Eur. Conf. on AI*, pages 359–379, 1992.
33. A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments. Technical report, 2001.
34. D. Le Berre and L. Simon. The essentials of the SAT'03 competition. In *SAT 2003*, 2003.
35. S. Malik, A. Wang, R. Brayton, and A. Sangiovanni Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *ICCAD*, 1988.
36. S. Minato. *Binary Decision Diagrams and Applications to VLSI CAD*. Kluwer, 1996.
37. D. B. Motter and I. L. Markov. A compressed breadth-first search for satisfiability. In *LNCS 2409*, pages 29–42, 2002.
38. R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *Proc. of IEEE/ACM Int. Workshop on Logic Synthesis*, 1995.
39. A. San Miguel Aguirre and M. Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *Principles and Practice of Constraint Programming*, pages 121–136, 2001.
40. T. Schaefer. The complexity of satisfiability problems. In *STOC'78*, pages 216–226, 1978.
41. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. 81(1-2):17–29, 1996.
42. F. Somenzi. CUDD: CU decision diagram package, 1998.
43. R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to tests chordality of graphs, tests acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.
44. T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *1st Int. Conf. on Constraints in Computational Logics*, pages 34–49, 1994.
45. A. Urquhart. The complexity of propositional proofs. *the Bulletin of Symbolic Logic*, 1:425–467, 1995.
46. L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *CAV 2002*, pages 17–36, 2002.