Symbolic Decision Procedures for QBF*

Guoqiang Pan, Moshe Y. Vardi

Dept. of Computer Science, Rice University {gqpan,vardi}@cs.rice.edu

Abstract. Much recent work has gone into adapting techniques that were originally developed for SAT solving to QBF solving. In particular, QBF solvers are often based on SAT solvers. Most competitive QBF solvers are search-based. In this work we explore an alternative approach to QBF solving, based on symbolic quantifier elimination. We extend some recent symbolic approaches for SAT solving to symbolic QBF solving, using various decision-diagram formalisms such as OBDDs and ZDDs. In both approaches, QBF formulas are solved by eliminating all their quantifiers. Our first solver, QMRES, maintains a set of clauses represented by a ZDD and eliminates quantifiers via multi-resolution. Our second solver, QBDD, maintains a set of OBDDs, and eliminate quantifier by applying them to the underlying OBDDs. We compare our symbolic solvers to several competitive search-based solvers. We show that QBDD is not competitive, but QMRES compares favorably with search-based solvers on various benchmarks consisting of non-random formulas.

1 Introduction

Propositional satisfiability (known as *SAT*) testing is one of the central problem in computer science; it is a fundamental problem in automated reasoning [44] and a key problem in computational complexity [16]. More recently, SAT solving has also shown to be effective in providing a generic problem-solving framework, with applications to planning [37], scheduling [18], bounded model checking [6], and more. Starting with the seminal papers [21, 22] in the early 1960s, the field has seen tremendous progress. Most SAT solvers today are based on the basic search-based approach of [21], rather than the resolution-based approach of [22]. Recently, highly tuned search-based SAT solvers [32, 57] have been developed, combining intelligent branching, efficient Boolean constraint propagation, backjumping, and conflict-driven learning. These solvers have shown to be quite effective in solving industrial-scale problems [17].

Quantified propositional satisfiability (known as *QBF*) captures problems of higher complexity (PSPACE vs NP), including temporal reasoning [51], planning [49], and modal satisfiability [46]. Much recent work has gone into adapting techniques that were originally developed for SAT solving to QBF solving, cf. [9, 41]. In particular, QBF solvers are often based on SAT solvers; for example, QuBE [31] is based on SIM [30], while Quaffle [58] is based on ZChaff [57]. Essentially all competitive QBF solvers are search-based [40]. In spite of the growing sophistication of QBF solvers, it is fair to say that they have shown nowhere near the effectiveness of SAT solvers [40].

* 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 explore an alternative approach to QBF solving, based on symbolic quantifier elimination. The underlying motivation is the success of symbolic techniques based on *binary decision diagrams* (BDDs) [8] and their variants in various automated-reasoning applications, such as model checking [10], planning [14], and modal satisfiability testing [45, 46]. Early attempts to apply symbolic techniques to SAT solving simply used the capacity of BDDs to represent the set of all satisfying assignments and were not too effective [56]. More recent efforts focused on SAT solving using quantifier elimination, which, in essence, goes back to the original approach of [22], since resolution as used there can be viewed as a variable-elimination technique, ala Fourier-Motzkin. (Resolution is typically thought of as a constraint-propagation technique [24], but since a variable can be eliminated once all resolutions on it have been performed [22], it can also be thought as a quantifier-elimination technique.) In [13] it is shown how *zero-suppressed decision diagrams* (ZDDs) [42] can offer a compact representation for sets of clauses and can support symbolic resolution (called there *multiresolution*). In [47, 50] it is shown how *ordered Boolean decision diagrams* (OBDDs) can support symbolic quantifier elimination. In both [13] and [47] the symbolic approach is compared to search-based approaches, showing that, search-based techniques seem to be generally superior, but the symbolic techniques are superior for certain classes of formulas.[1]

While the case for symbolic techniques in SAT solving cannot be said to be too strong, they are intriguing enough to justify investigating their applicability to QBF. On one hand, extending search-based technique to QBF has not, as we noted, been too successful. On the other hand, symbolic quantifier elimination handles universal quantifiers just as easily (and sometimes more easily) as it handles existential quantifiers, so extending symbolic techniques to QBF is quite natural. (Symbolic techniques have already been used to address conformant-planning problems [14], which can be expressed as QBF instances of low alternation depth.) In this work we investigate the two symbolic techniques to QBF. We extend the ZDD-based multi-resolution approach of [13] and the OBDD-based approach of symbolic quantifier elimination of [47]. We call the two approaches *QMRES* and *QBDD*. We compare these two approaches with three leading search-based QBF solvers: Quaffle and QuBE, which were mentioned earlier, and Semprop [49]. Unlike other comparative works [40], we decided to split our benchmark suite according to the provenance of the benchmarks, as our goal is to identify classes of problems for which the symbolic approaches are suited. We use a benchmark suite generated by Rintanen [49], which consists of a variety of constructed formulas (we omitted the random formulas), a second generated by Ayari [3], which consists of scalable formulas converted from circuit descriptions and protocol descriptions, and those generated by Pan [46], which consist of QBF formulas translated from modal logic formulas. Our experiments reveal that QMRES is significantly superior to QBDD. In fact, QBDD does not seem to be a competitive solver. (Though we return to this point at our concluding discussion.) In contrast, QMRES is quite competitive. While it is comparable to search-based method on Rintanen's formulas, QMRES outperforms them on Ayari's and Pan's formulas. At the same time, QMRES performs abysmally on random formulas. This suggests that symbolic techniques ought to be considered as comple-

[1] See `www.cs.rice.edu/~vardi/papers/`

mentary to search-based techniques and should belong in the standard tool kit of QBF solver implementors.

We start this paper with a description of current symbolic algorithms and the semantics of QBF in Section 2. We then describe our two symbolic QBF decision procedures in Section 3. We compare these solvers to search-based solvers in Section 4. We conclude with a discussion in Section 5.

2 Background

2.1 Symbolic Approaches to SAT

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 (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 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). The *support set* of an OBDD is the set of variables labeling its internal nodes.

In [56, 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 assignment to its variables, A_i is of linear size.) An OBDD for the set of satisfying truth assignments 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 in constant time, it makes little sense, however, to apply existential quantification to B_m. Suppose 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 [50]: 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 [33] (under the name of *hiding functions*) and tried on random 3-SAT instances. The work in [50, 47] studied this method further, and considered various optimizations.[2]

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 intermediate OBDDs. This observation was first made in the context of symbolic model checking, cf. [11, 29, 36, 7]. 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 [48] 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. A particular clustering heuristic was proposed in [25] in the context of constraint satisfaction. 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 minimum 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 processed in order of increasing rank. This approach is referred to as *bucket elimination* (BE) [23] (each cluster is thought as a "bucket") or as *adaptive consistency* [24]. An equivalent way of viewing this is to say that variable are eliminated in order of increasing rank, where eliminating a variable x requires conjoining of all OBDDs with x in their support set and then quantifying this variable existentially.

A good variable order for bucket elimination is an order that would minimize the size of the support set of intermediate OBDDs. The goal is to approach the *induced width*, which is the maximal support set under the optimal elimination order. Induced width is known to be equal to the *treewidth* [25, 28]. The treewidth of the formula is defined with respect to its *Gaifman graph*, whose vertices are the set of variables in the

[2] 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. [4].

formula and its edges connect variables that co-occur in a clause. (We use vertices and variables interchangeably.) Treewidth measures how close the graph is to being a tree; the treewidth of a tree is 1 [27]. BE, with respect to an optimal order, yields the optimal reduction of support set size [19] and is guaranteed to have polynomial running time for input instances of logarithmic treewidth, since this guarantees a polynomial upper bound on OBDD size.

Finding an optimal variable order for BE is known to be NP-hard, since computing the treewidth of a graph is NP-hard [2], so one has to resort to various heuristics, cf. [38]. An order that is often used in constraint satisfaction [24] and works quite well in the context of symbolic satisfiability solving [47] is the "maximum cardinality search" (MCS) order of [55], which is based on the graph-theoretic structure of the formula, i.e., the Gaifman graph. MCS ranks the vertices from n to 1 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).

If constraints are represented by clauses rather than by OBDDs, then variables can be eliminated via resolution. Given a set C of clauses and a variable x, the variable can be eliminated by adding to C all resolvents on x and then eliminating all clauses where x occurs. Formally $(\exists x)C$ is logically equivalent to $Resolve_x(C)$, where $Resolve_x(C)$ is the set of clauses obtained from C by adding all resolvents on x and then deleting all clauses containing x. In fact, completeness of resolution is shown by eliminating all variables one by one, each time replacing a set C of clauses by $Resolve_x(C)$ [22]. (Eliminating variables in such a fashion is reminiscent of Fourier-Motzkin variable elimination for systems of linear inequalities and of Gaussian variable elimination for systems of linear equalities.) This approach is also referred to as *directional resolution* [26] (see report there on experimental comparison of directional resolution to search-based techniques).

Rather than represent clauses explicitly as in [22, 26], multi-resolution [13] takes a symbolic approach to directional resolution, where clause sets are represented by ZDDs [42]. Each propositional literal ℓ is represented by a ZDD variable v_ℓ, 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 can 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 [42].

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 $c' \subset c$, where $c' \in C + D$. Multi-resolution can be easily implemented using $\times$ on cofactors: given a ZDD f, f_{x^+} and f_{x^-} is used to represent the ZDDs corresponding to the positive cofactor on the ZDD variable corresponding

to the literal $x/\neg x$, 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 with $f_{x'}$, which is the ZDD for the clauses not containing x.

2.2 QBF

Quantified Boolean Formulas (QBF) extend propositional logic by adding propositional quantifiers to propositions. In QBF terms, propositional satisfiability is the special case where all variables are quantified existentially. The addition of alternating quantifiers pushes the complexity of the problem from NP-complete for propositional satisfiability to PSPACE-complete for QBF [53].

We consider QBF formulas in prenex clausal normal form (CNF):

$$\varphi = Q_1 X_1 Q_2 X_2 \ldots Q_n X_n \varphi',$$

where each Q_j is $\forall$ or $\exists$, the X_is are disjoint sets of propositional variables, and φ' is a propositional formula in CNF. We refer to $Q_1 X_1 \ldots Q_n X_n$ as the *prefix* of φ and to φ' as the *matrix* of φ. We define A_φ as the set of universally quantified variables and, correspondingly, define E_φ as the set of existentially quantified variables. The *alternation depth* $alt(x)$ of a variable x is the index i where $x \in X_i$. All QBF formulas can be converted to prenex normal form with only a linear blow-up. We assume without loss of generality that all variables are quantified.

The semantics for QBF can be defined in terms of the semantics of propositional logic. If φ is quantifier free, then satisfaction ($\models$) is defined as for propositional logic. Otherwise, given an assignment α, we have that $\alpha \models \exists x\varphi$ iff $\alpha[x \mapsto \top] \models \varphi$ or $\alpha[x \mapsto \bot] \models \varphi$, and $\alpha \models \forall x\varphi$ iff $\alpha[x \mapsto \top] \models \varphi$ and $\alpha[x \mapsto \bot] \models \varphi$. If every variable in φ is quantified, then either for all assignments α, we have $\alpha \models \varphi$, or for all assignments α, we have $\alpha \not\models \varphi$. Thus, we can write $\models \varphi$ or $\not\models \varphi$; that is, satisfiability and validity coincide.

Most SAT solvers are search-based, following the ideas of [21]. QBF solvers build upon search techniques developed for SAT, forcing backtracking on universal variables and branching on variables according to alternation order [12]. A decision procedure based on an extension of resolution to QBF (called *Q-resolution*) is described in [9], but, to the best of our knowledge has not been implemented. We comment later on the difference between Q-resolution and our multi-resolution approach to QBF.

3 Symbolic Quantifier Elimination for QBF

The basic idea of our approach is to extend symbolic quantifier elimination from SAT to QBF. Given a QBF formula $\varphi = Q_1 X_1 Q_2 X_2 \ldots Q_n X_n \varphi'$, we eliminate the quantifiers from *inside out*, that is, in order of decreasing alternating depth, starting with the variables in X_n. At each stage, we maintain a set of constraints, represented symbolically either as a ZDD (expressing a set of clauses) or as a set of OBDDs. To eliminate an existential variable from a set of clauses we perform multi-resolution, while universal variables can be eliminated by simply deleting them [9]. To eliminate an existential variable x from a set of OBDDs we conjoin the OBDDs in whose support

set x occurs and then quantify it existentially, while to eliminate a universal variable we quantify it universally. (We can apply universal quantification to an OBDD B: $(\forall x)B = \text{APPLY}(B|_{x\leftarrow 1}, B|_{x\leftarrow 0}, \wedge)$.) The variables within a quantifier block $Q_i X_i$ are unordered and can be eliminated in any order. Here we apply the heuristics described in Section 2.1.

We note that our resolution approach to QBF is different than that of [9]. We require that quantifiers be eliminated from the inside out; thus, resolution can be performed only on the existential variables in the innermost quantifier block. In contrast, Q-resolution [9] allows resolution on non-innermost existential variables. The difference stems from the fact that the focus in Q-resolution is on generating resolvents, while the focus here is on quantifier elimination. For Q-resolution to be complete, *all* resolvents need to be kept. In contrast, once we have performed multi-resolution on a variable x, all clauses containing it are deleted.

First, we describe QMRES, a multi-resolution QBF solver. We provide pseudocode in Algorithm 1:

Algorithm 1 Multi-resolution for QBF

Q-Multi-Res(φ, S, v)
Require: S is the set of clauses forming matrix of φ, and $v = \langle v_1 \dots v_n \rangle$ is an order of variables where $alt(v_i) \geq alt(v_{i+1})$
Ensure: returns **true** if φ is valid and **false** otherwise
  for i=1..n **do**
    if v_i is existential **then**
      $S \Leftarrow (S_{v_i^+} \times S_{v_i^-}) + S_{v_i'}$
    else
      $S \Leftarrow S_{v_i^+} + S_{v_i^-} + S_{v_i'}$
    end if
    $S \Leftarrow Unitprop(S)$ {Apply unit propagation}
  end for
  return $S \neq \{\phi\}$

Note that in addition to multi-resolution the algorithm applies a naive form of unit propagation (weaker then what is described in [58]). When the clause set is represented using ZDDs, clauses with only a single literal can be easily enumerated without traversing the whole ZDD, since such clauses are represented by a path of length 2 in the ZDD. Existential unit literals can then be resolved on without regard to their alternation depth. (If a universal literal becomes unit, the formula is false.) The overhead of such a check are negligible so we applied it in all cases.

We now describe QBDD, an OBDD-based QBF solver. We provide pseudocode in Algorithm 2[3]:

In Section 2.1 we described the MCS heuristics for variable ordering. MCS is only one of many variable-ordering heuristics that are used to approximate treewidth [38].

[3] We used $choose\_bucket(R_i) = i + 1$, which avoided the overhead of traversing the BDD R_i and finding the lowest ordered variable according to v.

Algorithm 2 Bucket Elimination for QBF

QBDD(φ, S, v)
Require: S, v as in Q-Multi-Res
Ensure: returns true if φ is valid and false otherwise
Build BDD clusters $S_1 \ldots S_n$, where a clause $c \in S$ is in cluster S_i if v_i is the lowest ordered variable in c
for i=1..n **do**
if v_i is existential **then**
$R_i = \exists x_i \bigwedge_{c \in S_i} c$
else
$R_i = \forall x_i \bigwedge_{c \in S_i} c$
end if
if $R_i =$ **0 then**
return false
end if
$j = choose\_bucket(R_i)$
$S_j = S_j \cup \{R_i\}$
end for
return true

We explored several other variable-ordering heuristics in [47]. MCS came out as the overall best performers across a wide variety of benchmarks. It is interesting to note that MCS is not necessarily the best performer in terms of induced width. Other heuristics, such as *min-fill* [24] yield better induced width. One has to remember, however, that variable order impacts not only the induced width but also decision-diagram size. In turns out that a variable that reduce the size of the support set of decision diagram does not necessarily reduces its size. While these effects may be less marked for ZDD-based clause representation, we chose to use MCS for both of our algorithms.

As described earlier, however, MCS is computed from the matrix of the QBF formula, ignoring completely the quantifier prefix. Since quantifier elimination proceed from the inside out, we need to adapt MCS to take into account alternation depth. We first perform MCS on the matrix only, ignoring alternation. Then, variable order for quantifier elimination is generated, where at each step we choose a variable from those with the highest alternation depth that has the lowest MCS rank.

4 Experimental Results

We compare the symbolic approaches with three search-based solvers: QuBE [31], Quaffle [58], and Semprop [41]. These solvers use sophisticted heuristics for branch-variable selection and lemma/conflict caching. For both symbolic solvers, we used CUDD [52] as the underlying decision diagram engine, and for QMRES, we used the multi-resolution engine implemented by Chatalic and Simon [13].

We use three classes of benchmarks from the QBFLIB benchmark suites[43], those generated by Rintanen [49], from which we omitted the random formulas but kept a variety of hand constructed formulas, those generated by Ayari [3], which consist of

scalable formulas converted from circuit descriptions and protocol descriptions, and those generated by Pan [46], which consist of formulas translated from modal logic.

4.1 Symbolic vs. Search

A first observation, consistent with propositional logic, is that symbolic approaches typically performs very badly on random problems. For example, symbolic approaches are orders of magnitude slower for uniform propositional 3-CNF problems [47]. (In our QBF experiments, the symbolic approaches completed none of the uniform random formulas in the Rintanen's benchmarks within the 1000s timeout limit.) In the following, we compare the symbolic and search approaches only on constructed formulas, ignoring the results for random problems. (In general, QBF solvers typically behave quite differently on random vs. non-random formulas, which is why comparative studies separate the two cases [40].)

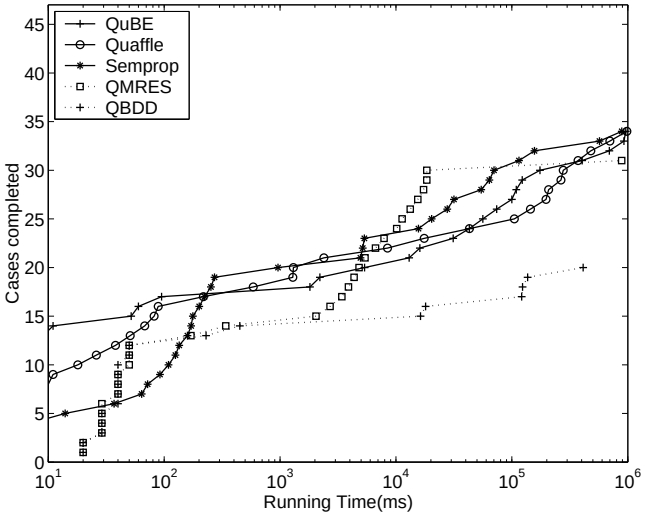

Fig. 1. Rintanen's Benchmarks (Non-random)

First we evaluated our solvers on Rintanen's and Ayari's benchmark suites [49, 3]. Rintanen's benchmark suite is one of the first benchmark suite for QBF, including formulas constructed from planning problems, hand-constructed formulas, and randomly generated formulas covering a wide range of difficulty and alternation depth. Ayari's benchmark suite is one of the first to encode bounded-model-construction problems in QBF through M2L-STR, a monadic second-order logic on finite words [3], with typically quite low alternation depth.

The results for Rintanen's benchmarks are plotted in Figure 1 and the results for Ayari's Benchmarks are plotted in Figure 2. We used the plotting style presented in [30, 54], which plotted the number of completed cases against the time taken to run each case. A solver with a higher curve dominates one with a lower curve since it solved more cases in the same amount of time.

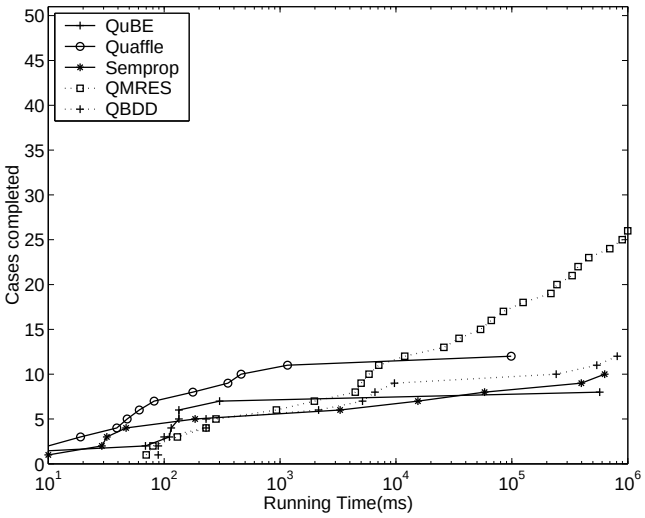

Fig. 2. Ayari's Benchmarks

Our first observation is that QMRES clearly dominates QBDD. This is somewhat surprising, since similar experiments we performed on propositional formulas showed that with the same variable order, the OBDD-based approach dominates the ZDD-based in most cases, falling behind only for highly under-constrained problems, where the compression of ZDD-based clause set representation is greater. In contrast, ZDD-based clause sets seems to be getting better compression across the range of the QBF problems, resulting in node usage that are orders of magnitude smaller then that of QBDD

Comparing symbolic solvers against search-based solvers, the picture is somewhat mixed. For Rintanen's benchmarks, there is no clear winner, but for Ayari's benchmarks, QMRES showed a clear edge. Some of the formulas in the Rintanen's benchmarks, for example, blocks problems, only have a low alternation depth and small number of universal variables, allowing search to perform effectively. In essence, such problems are closer to SAT, where search-based approach typically outperform symbolic solvers [47]. On the other hand, Ayari's problems are derived from circuit and protocol problems, whose symmetry favors the compression of the symbolic approach. It is interesting to note that the advantage of QMRES shows only when more difficult problems are considered. On easier problems search-based methods are faster.

Next, we come to formulas obtained by translation to QBF from modal formulas in the logic $\mathcal{K}$ [46]. The original modal formulas are scalable classes constructed by Heuerding and Schwendimann [35], where modal properties are nested to construct successively harder formulas. The resulting QBF formulas are grouped in the same 18 classes as the modal formulas, half satisfiable and half unsatisfiable, and each class contains 21 cases of which alternation depth scales linearly. Using translation from modal logic allowed construction of high alternation-depth problems that are both non-trivial and tractable. The formulas span a large range of difficulty and sizes, from hundreds to tens of thousands of variables. All the original modal formulas can be solved using modal solvers. We plotted time vs. cases solved in Figure 3. We see that QMRES clearly

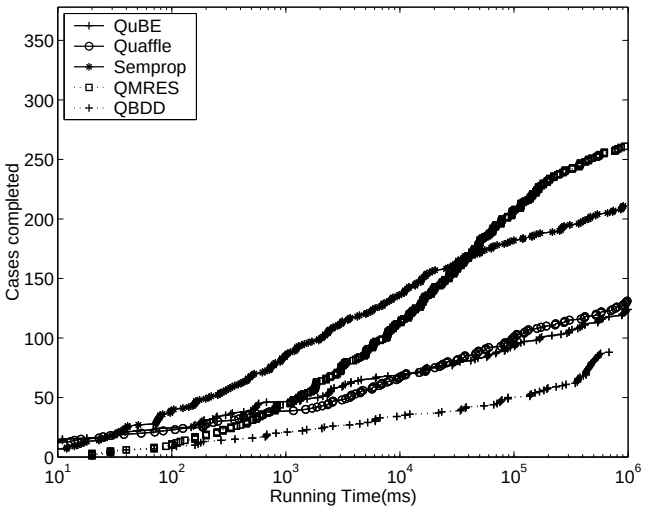

Fig. 3. Modal Logic Benchmarks

dominates the search-based methods. (This is consistent with the results described in [46], where symbolic modal solvers dominate search-based solvers.)

A fundamental question is why a symbolic method outperforms search-based method for QBF, while the situation is reversed for SAT [47]. A possible explanation is that propositional satisfiability is in essence a search for a satisfying assignment. SAT solvers excel in quickly eliminating unsuccessful search branches, combining back-jumping and conflict-based learning. In contrast, search-based QBF solvers have to deal with backtracking forced by the universal quantifers, which seems to nullify many of the advanced search heuristics of SAT solvers. Since QBF solving requires dealing with the whole space of possible assignments, symbolic methods benefit from the compression provided by decision diagrams.

4.2 QMRES vs QBDD

To better understand the disappointing performance of the BDD-based approach, we take a deeper look at Pan's formulas, which are generated from the modal formulas of [35] through two different translation steps, first from ML to non-CNF QBF, then from non-CNF QBF to QBF. In addition to running QMRES and QBDD on the final QBF formulas, we run KBDD, an OBDD-based modal solver [46], on the original modal formulas. We also run an ad-hoc OBDD-based solver on the intermediate non-CNF QBF formulas, where we translate propositional formulas to OBDDs without going through the CNF step and then apply quantifier elimination.

In Figure 4, we plotted the performance of the corresponding solvers. We see that the native solver KBDD performs best, with QMRES and QBDD-non-CNF very close to each other and not far behind the native solver. There is a much larger gap between these two and the performance of QBDD, resulting from the CNF translation. The gap between the performance of KBDD and QBDD-non-CNF can be attributed to the con-

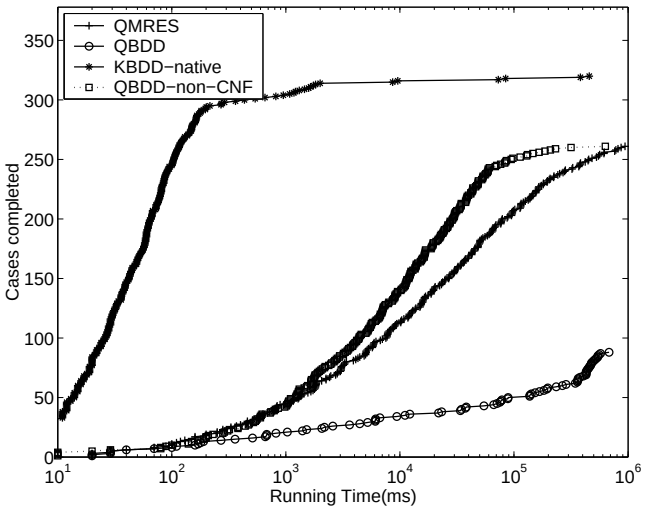

Fig. 4. Study of overhead on ML benchmarks

version of conjunction in modal formulas to universal quantification in the QBF translation. The gap between the performance of QMRES and QBDD-non-CNF to that of QBDD can be attributed to the the cost incurred by the OBDD-based solver in handling the additional variables generated in the conversion to CNF and the difficulty in implementing certain optimizations under the BDD representation, for example, unit propagation. suggests that the OBDD-based approach might be more effective for problems whose natural encoding is not in CNF.

5 Discussion

QBF, as a straightforward extension of propositional logic, shares much with SAT in the implementation of decision procedures. Nevertheless, alternation in QBF causes significant overhead to search-based solvers, requiring them to cover the whole search space. This greatly reduces the efficiency of the typical heuristics used for search-based solvers, whose optimization is usually focused toward minimizing the number of backtracks. For symbolic techniques, on the other hand, the addition of alternating quantifiers does not introduce analogous difficulties. Our results show that for difficult non-random QBF problems, symbolic quantifier elimination approaches are very competitive and belong in the standard tool kit of QBF solver implementors.

Recently, another quantifier elimination based solver (Quantor) was developed by Biere [5]. In addition to eliminating existential variables by resolution, universal variables can be eliminated using skolemization and expansion. His approach used a explicit representation of clauses instead of the symbolic representation used here. 38 previously unsolved constructed problems in the second QBF solver evaluation [39] have been solved using QMRES and Quantor. This bolsters support in arguing that quantifier elimination is an important addition to the pool of techniques of solver implementors.

Much further work is needed in this area, both to investigate various optimization techniques for symbolic solvers (e.g., variable order, non-CNF approaches, and representation beyond decision diagrams, for example, BEDs [1] or DNNFs [20]) as well as to investigate the trade-off between symbolic and search-based techniques. In particular, it would be interesting to investigate hybrid approaches, extending some work done in the context of model checking [34].

6 Acknowledgement

We would like to thank Laurent Simon for making available to us his ZRes package, which served as the basis for QMRES.

References

1. H. R. Anderson and H. Hulgaard. Boolean expression diagrams. *Informaton and Computation*, 179(2):194–212, 2002.
2. S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Math*, 8:277–284, 1987.
3. A. Ayari and D. Basin. Bounded model construction for monadic second-order logics. In *Proceedings of CAV'00*, 2000.
4. J. Balcazar. Self-reducibility. *Journal of Computer and System Sciences*, 41(3):367–388, 1990.
5. A. Biere. Resolve and expand. In *SAT 2004*, 2004.
6. A. Biere, Cimatti A, E.M. 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.
7. 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.
8. R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, Vol. C-35(8):677–691, August 1986.
9. H.K. Buning, M. Karpinski, and A. Flogel. Resolution for quantified Boolean formulas. *Inf. and Comp.*, 117(1):12–18, 1995.
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. J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *Int. Conf. on Very Large Scale Integration*, 1991.
12. M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
13. 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.
14. A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *J. of AI Research*, 13:305–338, 2000.
15. C. Coarfa, D. D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-SAT: The plot thickens. *Constraints*, pages 243–261, 2003.
16. S. A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

17. F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 2001.
18. J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI*, volume 2, pages 1092–1097, 1994.
19. V. Dalmau, P.G. Kolaitis, and M.Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP'02*, pages 310–326, 2002.
20. A. Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.
21. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.
22. S. Davis and M. Putnam. A computing procedure for quantification theory. *Journal of ACM*, 7:201–215, 1960.
23. R. Dechter. *Learning in graphical models*, chapter Bucket elimination: a unifying framework for probabilistic inference, pages 75–104. 1999.
24. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
25. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
26. 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.
27. R.G. Downey and M.R. Fellows. *Parametrized Complexity*. Springer-Verlag, 1999.
28. E.C Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. AAAI-90*, pages 4–9, 1990.
29. D. Geist and H. Beer. Efficient model checking by automated ordering of transition relation partitions. In *CAV 1994*, pages 299–310, 1994.
30. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. *Lecture Notes in Computer Science*, 2083, 2001.
31. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE, a system for deciding quantified Boolean formulae satisfiability. In *IJCAR'01*, 2001.
32. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver, 2002.
33. J. F. Groote. Hiding propositional constants in BDDs. *FMSD*, 8:91–96, 1996.
34. 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.
35. A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical report, Universität Bern, Switzerland, 1996.
36. R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. pages 12–19, 1996.
37. H. Kautz and B. Selman. Planning as satisfiability. In *Proc. Eur. Conf. on AI*, pages 359–379, 1992.
38. A.M.C.A. Koster, H.L. Bodlaender, and S.P.M. van Hoesel. Treewidth: Computational experiments. Technical report, 2001.
39. D. Le Berre, L. Simon, M. Narizzano, and A. Tacchella. QBF evaluation 2004. http://satlive.org/QBFEvaluation/2004/, 2004.
40. D. Le Berre, L. Simon, and A. Tacchella. Challenges in the qbf arena: the sat'03 evaluation of qbf solvers. In *In Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, 2003.
41. R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *TABLEAUX 2002*, pages 160–175, 2002.
42. S. Minato. *Binary Decision Diagrams and Applications to VLSI CAD*. Kluwer, 1996.

43. M. Narizzano. QBFLIB, the quantified Boolean formulas satisfiability library. http://www.qbflib.org.
44. A. Newell and H. A. Simon. The logic theory machine: A complex information processing system. *IRE Trans. Inf. Theory*, IT-2:61–79, 1956.
45. G. Pan, U. Sattler, and M.Y. Vardi. BDD-based decision procedures for K. In *Proc. of CADE 2002*, volume 2392 of *LNAI*, pages 16–30, 2002.
46. G. Pan and M.Y. Vardi. Optimizing a symbolic modal solver. In *Proc. of CADE 2003*, 2003.
47. G. Pan and M.Y. Vardi. Search vs. symbolic techniques in satisfiability solving. In *SAT 2004*, 2004.
48. 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.
49. J. Rintanen. Constructing conditional plans by a theorem-prover. *J. of A. I. Res.*, 10:323–352, 1999.
50. 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.
51. A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.
52. F. Somenzi. CUDD: CU decision diagram package, 1998.
53. L.J. Stockmeyer. The polynomial-time hierarchy. *Theo. Comp. Sci.*, 3:1–22, 1977.
54. G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial intelligence*, 131:39–54, 2001.
55. 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 Journal on Computing*, 13(3):566–579, 1984.
56. 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.
57. L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. In *CAV 2002*, pages 17–36, 2002.
58. L. Zhang and S. Malik. Towards symmetric treatment of conflicts and satisfaction in quantified Boolean satisfiability solver. In *CP'02*, 2002.