A Comparative Study of 2QBF Algorithms

Darsh Ranjan, Daijue Tang, and Sharad Malik

Princeton University
{dranjan,dtang,malik}@Princeton.EDU

Abstract. 2QBF is the problem of evaluating a Quantified Boolean Formula (QBF) with two levels of quantification. Many practical problems in sequential verification can be formulated as instances of 2QBF. Techniques that are not applicable to general QBF evaluation may be useful for 2QBF evaluation. In particular, decision order in search based algorithms may not obey quantification order for 2QBF evaluation algorithms. Different branching strategies in search based algorithms together with a resolution based method are described and compared. Experimental results on both random benchmarks and 2QBFs formulated from sequential circuit state space diameter problems are analyzed. Experiments show solvers specially tuned for 2QBF can be more efficient than similar general QBF solvers.

1 Introduction

The class of 2QBF problems is a subset of the class of Quantified Boolean Formulas (QBF), a generalization of Boolean satisfiability (SAT) problem. While SAT is known to be NP-complete, QBF is PSPACE-complete, and 2QBF is $\mathrm{NP}^{\mathrm{NP}}$-complete, so both 2QBF and QBF are likely to be much more difficult than SAT. Still, QBF attracts much research due to theoretical interest and practical applications such as artificial intelligence [8] and sequential circuit verification [9], [1].

The subclass 2QBF is worthy of study in its own right, away from the more general context of QBF. In particular, it may be useful to consider algorithms and techniques specific to 2QBF that may not generalize easily to QBF, as is done in this paper. Moreover, since 2QBF is a gentler generalization of SAT than general QBF, techniques that are useful in SAT algorithms sometimes adapt more easily and more usefully to 2QBF than they do to QBF. In fact, all three of the algorithms we discuss are strongly related to algorithms for propositional satisfiability. This is an advantage, since the study and development of SAT algorithms is more mature than the study of QBF algorithms. In this paper, we study three 2QBF algorithms. Two of them are based on the DPLL procedure [4] and are engineered toward solving 2QBF instances. The third is a resolution algorithm that is not specially designed for 2QBF; it is presented here because it has not been studied separately for 2QBF and because it is useful as a point of comparison.

2 Preliminaries

2.1 Definitions

Let ϕ be a Boolean formula. In this paper, we will assume that ϕ is a formula in conjunctive normal form (CNF) (other formulas may be converted to CNF by first writing a circuit for the formula and then expressing the circuit in CNF). If $x_1, x_2, \ldots, x_n$ are free variables of ϕ, then $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi$ is a quantified Boolean formula, where each Q_i is a quantifier $\forall$ or $\exists$. By the semantics of quantification, we can regard quantifiers as quantifying sets of variables rather than variables themselves, and then it suffices to require that quantifiers alternate between $\forall$ and $\exists$ in the prefix.

In general, as defined in [3], kQBF is the subclass of QBF consisting of such formulas with k quantifiers, so in particular, a 2QBF formula has two quantifiers. If ϕ is in CNF, then it is easy to see that a formula $\exists Y \forall X \phi$ simplifies to the SAT problem $\exists Y \phi'$, where ϕ' is the CNF obtained from ϕ by removing all occurrences of universal literals, so such formulas need not concern us in our study of 2QBF; we will only deal with 2QBF formulas of form $\forall X \exists Y \phi$.

2.2 Practical 2QBF from Sequential Verification

The algorithms described in this paper were tested on both randomly generated 2QBF problems and structured circuit 2QBF instances arising from the state space diameter problem for sequential circuits, which asks for the maximum over all ordered pairs of states (R, T) of the minimum length of a path from R to T (i.e., the *diameter* of the state machine). [10] illustrates how this problem may be formulated as a 2QBF. In [10], it is also argued that search methods, like some of the methods presented in this paper, ultimately cannot be effective in solving diameter problems. However, circuit 2QBF's arising from the diameter problem still provide a testing ground for the algorithms.

3 2QBF Algorithms

The remainder of this paper will be in the context of the 2QBF formula $F = \forall X \exists Y \phi$, where ϕ is in CNF.

3.1 DPLL-related algorithms

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm was originally described by Davis, Logeman, and Loveland in [4] as a decision procedure for SAT, but since [3] it has also been adapted widely for solving QBF instances. In this section, we present two algorithms based on DPLL—in fact, that are built on DPLL SAT solvers—for solving 2QBF. In both cases, the ZChaff SAT solver was used as the basis of the 2QBF solver, and the efficient incremental SAT capabilities of ZChaff were used.

Algorithm I. Algorithm A uses two DPLL SAT solvers, solver A and solver B, that communicate information to each other. Solver A maintains a CNF ϕ_{A} which begins as the original CNF ϕ, and solver B maintains a CNF ϕ_{B} which begins empty and is incremented during the solving process to contain clauses that depend only on the universal variables.

1. Solver A begins by finding a satisfying assignment α for ϕ.
   (a) If none exists, then the algorithm halts and returns *false*.
   (b) Otherwise, the solver then finds a cover set α' of α, a partial assignment of α that also satisfies all the clauses of ϕ. The complement of the conjunction of universal literals in α' is added as a clause b_1 to ϕ_{B}.
2. Solver B then finds a satisfying assignment β of the variables in X for ϕ_{B}, provided one exists.
   (a) If it does not exist, then the algorithm halts and returns *true*.
   (b) If β exists, on the other hand, then β is taken as an initial assignment for solver A, and we return to step 1, where solver A tries to find a satisfying assignment of ϕ_{A} given β as an initial assignment, and the process repeats.

Given any initial assignment to X, solver A will find an extension of it that satisfies ϕ if one exists, so if given some initial assignment, solver A fails to find a satisfying assignment, then F itself must be *false*. On the other hand, by induction, at every point in the solving process, ϕ_{B} will have the property that any universal assignment failing to satisfy it is known to have corresponding satisfying existential assignments, so that if solver B ever fails to find a satisfying assignment, then the space of all universal assignments has been covered, and F must be *true*. Finally, note that after every iteration on which both solvers find satisfying assignments, at least one satisfying assignment is removed from ϕ_{B}, so the behavior of both solvers successfully and repeatedly finding satisfying assignments cannot continue forever. At some point either A or B will fail to find a satisfying assignment, at which point F will be evaluated. This shows that Algorithm I is a sound and complete 2QBF algorithm.

Remark 1. Conceptually, this algorithm is very similar to the Quaffle algorithm (described in [11]) restricted to 2QBF. The primary difference is that focusing our attention to 2QBF allows us to rely on SAT solvers with efficient data structures like 2-literal watching, unlike Quaffle.

The implementation of algorithm I also includes a preprocessing stage in which the trivial truth and falsity of the formula is checked. The trivial truth rule is that if the CNF is satisfiable if all the universal literals are ignored, then the formula itself is *true*. The trivial falsity rule is that if the set of clauses containing only existential variables is unsatisfiable, then the formula itself if *false*. These are discussed in more detail in [3].

Algorithm II. Algorithm II is a variation of algorithm I. As before, there are two SAT solvers, A and B, that have the same tasks as in algorithm I. The difference is that in algorithm II, when solver B finds a satisfying assignment to X, solver A does not use that assignment itself as an initial assignment; solver A finds its own satisfying assignment independently of solver B's assignment, i.e., in algorithm II, step 1(b) is replaced by the following two steps:

1. (b) i. Solver A finds a cover set α' of α, a partial assignment of α that also satisfies all the clauses of ϕ. The complement of the conjunction of universal literals in α' is added as a clause b_1 to ϕ_B.
   ii. Solver A then finds a clause b_1' that has the property that $\phi_\mathrm{A} \wedge b_1$ is logically equivalent to $\phi_\mathrm{A} \wedge b_1'$ (this is done by taking a cutset of the implication graph of the complements of all of the literals in b_1; implication graphs are explained in [12]). ϕ_A is then augmented by b_1'.

Also, in step 2(b), β is not taken as an initial assignment for solver A.

NOTE: unlike in algorithm I, the satisfying assignment found by solver B is not used by solver A in any way. However, the clause added to ϕ_A in step 1(b).ii adds enough information to ϕ_A so that the satisfying assignment (or lack of one) can be treated the same way, and the proof of correctness of algorithm I can be applied with slight modification to algorithm II. Experimental results using different heuristics for obtaining b_1' from b_1 have been obtained (the heuristics tested are described in more detail in section 4).

Another important aspect of algorithm II is that solver A can make use of the quantification level of its variables in its decision heuristic. This is described in more detail in section 4.

The implementation of algorithm II includes the same trivial truth and falsity preprocessing checks as algorithm I, but they were disabled in order to get more meaningful data from the randomly generated benchmarks (see section 4 below).

In both algorithms I and II, solver B uses the standard ZChaff incremental SAT engine.

Noncritical signals in Circuit 2QBF. Both algorithms I and II rely heavily on the notion of a cover set of a satisfying assignment α of a CNF formula. As explained above, the cover set α' is a subset of α such that each clause in the CNF is satisfied by an assignment in α'. The relevant fact about a cover formulated this way, though, is that for any assignment of all of the universal variables that contains all the universal assignments in α', there is an assignment of the existential variables that makes ϕ *true*. We will refer to the universal part of such a cover, which is what actually needs to be found in order to proceed, as a *universal cover*. In algorithms I and II, as long as the universal cover has this property, the algorithm will be correct, even if some of the clauses are not explicitly satisfied. In the 2QBF's studied arising from circuits, all universal variables are primary inputs. It may be clear from the circuit representation, though, that some subset of the primary inputs is itself sufficient to guarantee that the output will be *true*; in this case, the other primary inputs need not be considered. [10] explains this type of reasoning in more detail. Noncritical signal reasoning with AND and OR gates can potentially lead to better universal covers, and algorithms I and II were both tested on circuit problems with and without noncritical signal reasoning.

Remark 2. Although both the original SAT DPLL and the QBF extension [3] are polynomial space algorithms, both algorithms I and II, though based on DPLL, are not polynomial space algorithms. The clauses added to ϕ_B when solver A finds satisfying leaves of ϕ_A must be kept, and there is nothing that keeps ϕ_B from growing to exponential size. In some cases, it might even be necessary for ϕ_B to grow to order exponential in the size of the input before the formula is solved.

3.2 2QBF Resolution

The SAT resolution algorithm generalizes easily to both general QBF and 2QBF. The general algorithm, Q-resolution, is treated in [2]; only the 2QBF case, which is the restriction of Q-resolution to 2QBF, is discussed here. The algorithm described here is not specially tuned to solving 2QBF in any way. However, a variant of Q-resolution was implemented and tested along with vanilla Q-resolution. In this variant, before resolving an existential variable out of the formula, a set of clauses containing this variable is chosen and fed to the Espresso logic minimization tool, and the original formula is replaced by Espresso's simplification.

4 Experimental Results

4.1 Notes

These tests were done on an Intel Pentium III 933 MHz machine with 1 GB of RAM running Linux. Code was written in C++ and compiled using g++ 2.96 with the -O3 option.

Different variants of the algorithms described above were tested. Algorithms I and II were tested on circuit benchmarks both with and without the noncritical signal reasoning outlined above (In the tables, the presence of a plus sign (+) indicates augmention by noncritical signal reasoning). Algorithm II was tested using different branch variable selection strategies: variable state independent decaying sum ("VSIDS," or "V") is the standard strategy of the ZChaff solver (see [7]); "existential preferred" ("EP") is a similar strategy that gives existential variables added weight, and "existential first" (E1) is a similar strategy that always branches on free existential variables before branching on universal variables. Algorithm II was also tested using different "conflict analysis" techniques for obtaining clauses to add to solver A from covering clauses added to solver B (hereafter referred to as the "pruning strategy"): "decisions only" ("DO") and "first unit implication point" ("first UIP," or "1U") (see [12]). The resolution algorithm was tested on two settings: in "S", the formula is simplified between resolution steps, whereas in "NS", it is not. In this section we also compare these solvers with the general QBF solver Quaffle [11], since the DPLL-related algorithms treated here are closely related to it. We also compare these solvers with the Semprop QBF solver of Letz [5], which was judged the best overall QBF solver in the SAT 2003 QBF evaluation, for reference.

In all tests, each solver was limited to 400 seconds of CPU time.

4.2 Random Benchmarks

Table 1. Average runtime on random benchmarks

Number of clauses	Alg. I	Alg. II						Res.		Quaffle	Semprop
		E1,1U	E1,DO	EP,1U	EP,DO	V,1U	V,DO	NS	S		
100 (100 instances)	*0.01(100)	12.57(99)	5.85(100)	16.21(98)	2.06(100)	16.22(98)	2.06(100)	0.01(100)	0.25(100)	400.00(0)	0.01(100)
150 (10 instances)	*0.01(10)	400.00(0)	400.00(0)	400.00(0)	400.00(0)	400.00(0)	400.00(0)	0.01(10)	0.31(10)	400.00(0)	*0.01(10)
200 (10 instances)	160.00(6)	400.00(0)	400.00(0)	400.00(0)	400.00(0)	400.00(0)	400.00(0)	0.36(10)	169.09(6)	400.00(0)	23.99(10)

The random benchmarks all have approximately 100 variables, with five variables per clause. When a solver failed to solve an instance in 400 seconds, the runtime was taken to be 400 seconds. A number in parentheses indicates the number of instances the solver was able to solve. "*0.01" in the tables means that the time was beneath the granularity of the timing function, so the time is estimated at 0.01. (Most of the instances were trivially true, which was immediately detected by the preprocessing stage of algorithm I. In order to obtain useful data on these problems, the trivial truth preprocessing of algorithm II was disabled.)

4.3 Circuit Benchmarks

In these tables, the presence of a dash as the runtime for a particular instance means that the solver could not solve it in 400 seconds.

Algorithm I (+) was also tested on s1423 unrolled to 34 frames (determined to be false in 298.86 seconds) and 35 frames (not solved in 400 seconds). This improves the known lower bound for the diameter of s1423, a known hard instance, from 26 [6] to 34.

Table 2. Runtime on diameter problem benchmarks (algorithms with no critical signal reasoning)

Circuit	frames un-rolled	Alg. I	Alg. II						Res.		Quaffle	Semprop
			E1,1U	E1,DO	EP,1U	EP,DO	V,1U	V,DO	NS	S		
s27	1	0.03	0.03	0.02	0.03	0.03	0.03	0.03	0.03	0.14	0.01	0.01
	2	0.35	0.20	0.33	0.16	0.33	0.15	0.32	0.02	0.17	0.20	0.04
	3	25.68	4.98	12.00	3.02	8.58	3.03	8.62	0.04	0.33	8.58	0.20
	4	–	120.31	–	68.28	–	68.85	–	0.04	0.39	–	5.81
	5	–	–	–	–	–	–	–	0.17	0.40	–	21.29
s1488	1	0.06	26.56	25.20	24.50	24.85	24.30	24.34	–	2.19	0.27	0.25
	2	0.22	–	–	–	–	–	–	–	11.85	1.54	1.12
	3	2.25	–	–	–	–	–	–	–	25.04	18.23	30.75
	4	4.14	–	–	–	–	–	–	–	120.34	–	–
	5	65.49	–	–	–	–	–	–	–	208.60	–	–
s1423	1	0.27	–	–	–	–	–	–	–	4.29	0.22	0.24
	3	0.64	–	–	–	–	–	–	–	–	–	–
	6	1.20	–	–	–	–	–	–	–	–	–	–
	10	2.05	–	–	–	–	–	–	–	–	–	–
	15	3.91	–	–	–	–	–	–	–	–	–	–
	21	136.91	–	–	–	–	–	–	–	–	–	–

Table 3. Runtime on diameter problem benchmarks (algorithms with critical signal reasoning)

Circuit	frames un-rolled	Alg. I	Alg. II					
		(+)	E1,1U(+)	E1,DO(+)	EP,1U(+)	EP,DO(+)	V,1U(+)	V,DO(+)
s27	1	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	2	0.02	0.02	0.04	0.02	0.03	0.02	0.04
	3	0.08	0.05	0.15	0.03	0.12	0.03	0.13
	4	0.23	0.19	0.34	0.10	0.32	0.10	0.32
	5	0.92	0.62	8.72	0.31	8.67	0.32	8.63
s1488	1	0.15	0.26	0.33	0.24	0.23	0.24	0.23
	2	0.25	4.97	18.18	4.21	5.95	4.16	5.66
	3	0.80	294.27	–	80.38	–	79.43	–
	4	10.07	–	–	–	–	–	–
	5	2.50	–	–	–	–	–	–
s1423	1	0.15	20.70	31.01	17.16	36.33	17.47	36.41
	3	0.37	–	–	–	–	–	–
	6	0.70	–	–	–	–	–	–
	10	1.94	–	–	–	–	–	–
	15	2.11	–	–	–	–	–	–
	21	9.19	–	–	–	–	–	–

5 Discussion

A glance at the tables shows that overall, algorithm I is the most robust on these classes of problems, being able to solve the most instances (although resolution and Semprop were better for the random problems, which were relatively small).

For the random problems, although most of the instances could be solved almost instantly by trivial truth preprocessing (as algorithm I does), they seem still to be difficult to solve by other means. Incorporation of dynamic trivial truth in DPLL-based solvers is an area of current research.

For the circuit problems, as one would expect, the exploitation of noncritical circuit signals (which are not apparent in the CNF representation) in the DPLL-based solvers improved performance quite dramatically, often by several orders of magnitude. These results show that at the very least, it can be fruitful to hold on to some information from the original representation while working with the CNF counterpart.

The resolution algorithms compare favorably against all of the other solvers on most of the smaller problems. However, as one would expect, as the problems get larger (e.g., the s1488 and s1423 circuit benchmarks), their performance drops. The simplification steps predictably seem to pay off more on large problems, where the overhead is less significant compared to the overall solving time.

In algorithm II, the difference between the EP and plain VSIDS strategies may be attributed to noise. The E1 strategy shows a clear overall performance loss on the circuit benchmarks while being comparable to VSIDS on the random benchmarks. This can be explained by the fact that on the random benchmarks, the satisfying existential assignments do not depend strongly on the universal variables, while the circuit benchmarks do not have this property, so forcing existential decisions first is not fruitful.

This data does not clearly show either the DO or 1U pruning strategies to be superior to the other. Research centering on pruning strategies may be worthwhile.

Although Quaffle outperforms algorithm I on some of the smaller circuit problems, the overall trend is the opposite, showing that specialized 2QBF algorithms can be more efficient than their general QBF counterparts. Algorithm II clearly outperforms Quaffle on the random problems (of which Quaffle could not solve any), but on the circuit problems, Quaffle does better generally than the algorithm II variants. A more careful look shows that Quaffle typically does not explore nearly as many satisfying assignments as algorithm II, indicating that pruning the search tree more effectively, possibly by using a better pruning heuristic, may be a good area for improving algorithm II.

We note that algorithm I (especially algorithm I (+)) generally outperformed Semprop on the circuit problems, while Semprop was more effective in general on the random problems.

6 Conclusions and Future Work

In this paper, we have given strong evidence that focusing on 2QBF solvers as a class separate from general QBF solvers can be worthwhile. Also, for real world circuit problems, we have shown that using information from the circuit representation (outside the CNF representation) can improve performance.

There are many directions in which the algorithms described can be improved. The authors believe that a dynamic "trivial truth" check will improve the DPLL-like algorithms, especially in a streamlined, 2QBF-specific form; this is currently being studied.

References

1. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Proceedings of Tools and Algorithms of the Analysis and Construction of Systems (TACAS'99). (1999)
2. Büning, H.K., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Information and Computation **117** (1995) 12–18
3. Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. Journal of Automated Reasoning **28** (2002) 101–142
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM **5** (1962) 394–397
5. Letz, R.: Lemma and model caching in decision procedures for quantified Boolean formulas. In: Internation conference on Automated Reasoning with Analytics Tableaux and Related Methods (Tableaux 2002). (2002)
6. Mneimneh, M., Sakallah, K.: Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In: Sixth Internation Conference on Theory and Applications of Satisfiability Testing. (2003)
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the Design Automation Conference (DAC). (2001)
8. Rintanen, J.: Constructing conditional plans by a theorem prover. Journal of Artificial Intelligence Research **10** (1999) 323–352
9. Sheeran, M., Singh, S., Stålmark, G.: Checking safety properties using induction and a SAT-solver. In: Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design (FMCAD 2000). (2000)
10. Tang, D., Yu, Y., Ranjan, D., Malik, S.: Analysis of search based algorithms for satisfiability of quantified boolean formulas arising from circuit diameter problems. Submitted to The Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)
11. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In: Proceedings of 8th International Conference on Principles and Practice of Constraint Programming (CP2002). (2002)
12. Zhang, L., Madigan, C.F.,Moskewicz, M.H., Malik, S.: Efficient conflict driven learning in a Boolean satisfiability solver. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD). (2001)