

# A SAT-based Solver for Q-ALL SAT

B. Browning  
University of West Georgia  
Carrollton, Georgia, USA  
ben@westgaweb.com

A. Remshagen  
University of West Georgia  
Carrollton, Georgia, USA  
anja@westga.edu

## ABSTRACT

Although the satisfiability problem (SAT) is NP-complete, state-of-the-art solvers for SAT can solve instances that are considered to be very hard. Emerging applications demand to solve even more complex problems residing at the second or higher levels of the polynomial hierarchy. We identify such a problem, called Q-ALL SAT, that arises in a variety of applications. We have designed a solution algorithm for Q-ALL SAT that employs a SAT solver and thus exploits the recent advances of SAT solvers. In addition, a heuristic is applied to reduce the number of instances that are to be solved by the SAT solver. A learning scheme improves the performance of that heuristic. Test results of a first implementation of the proposed algorithm confirm that this is a very promising approach.

## Categories and Subject Descriptors

D.1.6 [Programming Techniques]: Logic Programming; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—*computational logic*; I.2.6 [Artificial Intelligence]: Learning

## General Terms

Algorithms

## Keywords

Logic programming, quantified Boolean formula, learning

## 1. INTRODUCTION

Research on quantified Boolean formulas has surged in recent years due to emerging applications that can be represented by quantified Boolean formulas. Such problems arise, for example, in medical diagnosis, robot navigation, planning problems, and regulatory compliance. See Eiter and Gottlob [2], Remshagen and Truemper [8], or Rintanen [9] for applications. Many of these problems can be represented by a quantified formula of the form

$$\forall Q (\exists X R \Rightarrow \exists Y S) \quad (1)$$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SE'06 March 10-12, 2006, Melbourne, Florida, USA  
Copyright 2006 ACM 1-59593-315-8/06/0004 ...\$5.00.

where  $R$  and  $S$  are propositional logic formulas in conjunctive normal form and where  $Q$ ,  $X$ , and  $Y$  are variable sets. We call the problem that demands to evaluate formula (1) *Q-ALL SAT*. Consider, for example, a planning problem where we want to determine a feasible plan that leads to a goal. In this case, the  $Q$  variables represent the different actions that can be chosen for a plan. Formula  $R$  models all feasible plans, and formula  $S$  models all scenarios in which a plan can fail. If formula (1) evaluates to *true*, then all feasible plans will fail. In other words, if formula (1) is *false*, there exists at least on feasible plan that does not fail, assuming  $R$  is satisfiable. Typically, unsatisfiability of  $R$ , that is, no feasible plan exists, does not occur in practical applications.

Q-ALL SAT is  $\Pi_2^P$ -complete and thus, considered to be even harder than NP-complete problems. Nevertheless, we believe that effective solution algorithms for practical applications can be developed. We have designed and implemented a Q-ALL SAT solver that reduces each instance to a sequence of problems which are at a lower complexity level than Q-ALL SAT. In particular, we have reduced the problem to a sequence of satisfiability problems (SAT) which demand to evaluate a propositional logic formula. Since SAT is well-known to be NP-complete, it is unlikely that SAT can be solved in polynomial time. However, state-of-the-art SAT solvers have experimentally proven to be very efficient (SAT 2005 [11]). Thus, the Q-ALL SAT solver takes advantage of effective SAT solvers. In addition, we apply a heuristic and a learning scheme to reduce the number of SAT instances to be solved.

The next section introduces terminology required to cover related work in Section 3 and to describe the solution algorithm in Section 4. Section 5 discusses computational results on two sets of benchmark problems. The last section summarizes the conclusions.

## 2. PRELIMINARIES

An instance of *Q-ALL SAT* contains two propositional logic formulas,  $R$  and  $S$ . The formulas  $R$  and  $S$  are in *conjunctive normal form*; that is, each formula is a conjunction of *clauses* where each clause is a disjunction of *literals*. A literal is a variable or a negated variable. The formulas  $R$  and  $S$  have a set of common variables, called  $Q$ . Formula  $R$  contains an additional variable set  $X$ , and  $S$  contains an additional variable set  $Y$ . We call a truth assignment to the  $Q$  variables *R-acceptable* if the assignment can be extended to an assignment to all variables of  $R$  so that  $R$  evaluates to *true*. An assignment to  $Q$  is *R-unacceptable* if  $R$  evaluates to *false* under every possible extension of that assignment. The terms *S-acceptable* and *S-unacceptable* are defined analogously. Q-ALL SAT demands to determine

whether all  $R$ -acceptable assignments to the  $Q$  variables are  $S$ -acceptable.

A well-known standard format for quantified Boolean formulas, called *QBF*, is as follows

$$\forall X_1 \exists X_2 \forall X_3 \dots \exists X_n S \quad (2)$$

or

$$\exists X_1 \forall X_2 \exists X_3 \dots \exists X_n S \quad (3)$$

where  $S$  is a propositional formula in conjunctive normal form and the variable set of  $S$  is  $X_1 \cup X_2 \cup \dots \cup X_n$ . Each of the formulas (2) and (3) has  $n - 1$  quantifier alterations. A QBF formula with  $n - 1$  quantifier alterations is at the  $n$ th complexity level of the polynomial hierarchy. See Papadimitriou [6] for the polynomial hierarchy. For example, the formula  $\forall X_1 \exists X_2 S$  has one quantifier alteration and therefore, is at the second level of the polynomial hierarchy. In particular, the formula is  $\Pi_2^P$ -complete. Hence, it belongs to the same complexity class as Q-ALL SAT. The subproblem of QBF that requires to evaluate formulas of the form  $\forall X_1 \exists X_2 S$  is called *2QBF*. Observe that, in the case of no quantifier alterations, formula (??) is reduced to the satisfiability problem (SAT).

### 3. PRIOR WORK

Q-ALL SAT has been introduced first in Truemper [13] who proposes a solution algorithm that reduces the problem to NP-hard minimization problems. Mneimneh and Sakallah [5] compute the eccentricity of vertices, a problem arising in hardware verification, by solving a sequence of Q-ALL SAT instances. Their solution algorithm is similar to the SAT-based solver described here. However, they do not apply a heuristic or learning process to reduce the number of SAT instances to be solved. Remshagen and Truemper [8] suggest a solution scheme based on backtracking-search. The problem Q-ALL SAT is closely related to logic-based abduction. Eiter and Gottlob [2] describe several applications and determine the complexity of logic-based abduction.

The problem Q-ALL SAT can be reduced to a 2QBF problem. However, that transformation increases the number of variables and the number of clauses of the involved CNF formulas significantly. Therefore, a transformation to QBF may not always be desirable; see Section 5 for details. Quantified Boolean formulas at the second level of the polynomial hierarchy have been investigated by Rintanen [9], who shows that conformant planning is  $\Pi_2^P$ -hard and translates conditional planning problems to instances of QBF. Ranjan, Tang, and Malik [7] compare the performance of QBF solvers and specialized 2QBF solvers on 2QBF instances. Their experiments indicate that specialized solvers for 2QBF can be more efficient than general QBF solvers.

Most QBF solvers for the general case are based on backtracking search, which has been proven to be very successful in solving the satisfiability problem SAT; see, for example, Zhang, Madigan, Moskewicz, and Malik [15]. Cadoli, Schaefer, Giovanardi, and Giovanardi [1] explore different heuristics with their solver Evaluate to choose the branching literal and investigate properties of randomly generated quantified Boolean formulas. Giunchiglia, Narrizano, and Tacchella [3], [4] implement a learning scheme in their solver QuBE. Rintanen [10] extends the Davis-Putnam procedure by unit propagation for quantified formulas. The QBF solver Quaffle by Zhang and Malik [17] learns conflict clauses dur-

ing the solution process. A satisfiability-driven learning scheme by Zhang and Malik [16] augments the CNF formula by a formula in disjunctive normal form.

## 4. A SAT-BASED SOLVER

We outline first a SAT-based solution algorithm for the Q-ALL SAT problem. Then we describe a heuristic that learns shortened clauses to reduce the search space. Finally, we present a learning scheme that improves the performance of the heuristic.

To evaluate formula (1), the algorithm first solves the SAT instance  $R$ . That is, it determines an assignment to the variables of  $R$  under which  $R$  evaluates to *true*. If no such assignment exists, formula (1) evaluates to *true*. Otherwise, the satisfying assignment contains an  $R$ -acceptable assignment to  $Q$ . We fix that  $Q$  assignment in CNF formula  $S$  and solve the resulting SAT instance. If the SAT instance has no solution, the  $Q$ -assignment is  $S$ -unacceptable. Thus, we have determined an  $R$ -acceptable and  $S$ -unacceptable assignment and hence, formula (1) evaluates to *false*. In the other case, the current  $Q$  assignment is  $S$ -acceptable. Now the algorithm determines another  $R$ -acceptable assignment and checks whether it is  $S$ -acceptable as well. In order to find another  $R$ -acceptable assignment, we cannot solve  $R$  again since a deterministic SAT solver will always obtain the same assignment. Therefore, we determine a clause that evaluates to *false* under the previous  $Q$  assignment and append that clause to  $R$ . Then, we begin the whole process over by solving  $R$  again. The new clauses guarantee that in every iteration a new  $R$ -acceptable  $Q$  assignment is found, until  $R$  becomes unsatisfiable. We outline the algorithm:

```

solve the SAT instance  $R$ ;
while ( $R$  is satisfiable)
  let  $\alpha$  be the satisfying truth assignment to  $Q$ ;
  solve the SAT instance resulting from fixing  $\alpha$  in  $S$ ;
  if ( $S$  is unsatisfiable)
    return false;
  add a clause to  $R$  that consists exactly of the literals  $v$ 
  where  $v$  is set to false by  $\alpha$  or  $\neg v$  is set to true;
  solve the SAT instance  $R$ ;
return true;

```

Essentially, we have broken the Q-ALL SAT problem down into a sequence of SAT problems. Extensive research has been conducted to design efficient SAT solvers. For example, see SAT 2005 [11]. Instead of developing our own SAT solver, our algorithm can take advantage of any existing SAT solver.

### 4.1 Learning Shortened Clauses

Observe that in each iteration of the above algorithm, we exclude only a single assignment to the  $Q$  variables from all possible  $R$ -acceptable assignments. We will use a heuristic that determines which  $Q$  variables do not have an impact on the satisfiability of formula  $S$ . That means, no matter which truth value is assigned to those  $Q$  variables, the resulting  $Q$  assignment is still  $S$ -acceptable. Assume, for example, the  $Q$  assignment  $q_1 = \text{true}$ ,  $q_2 = \text{true}$ ,  $q_3 = \text{false}$ , and  $q_4 = \text{false}$  is  $S$ -acceptable. Further assume that, if we remove all occurrences of  $q_2$  and  $q_4$  from the clauses of  $S$ , the partial assignment  $q_1 = \text{true}$ ,  $q_3 = \text{false}$  is still  $S$ -acceptable. That means, that no matter how we fix the variables  $q_2$  and  $q_4$ , the resulting  $Q$  assignment that sets  $q_1 = \text{true}$  and  $q_3 = \text{false}$  is  $S$ -acceptable. Hence we can exclude these assignments by adding the clause  $\neg q_1 \vee q_3$  to  $R$ .

To determine which  $Q$  variables are irrelevant for satisfiability of  $S$ , we apply the following heuristic whenever an  $R$ - and  $S$ -acceptable assignment to  $Q$  has been computed. We remove all occurrences of a  $Q$  variable, say  $q$ , from the clauses of  $S$  and solve the reduced CNF formula again. If the formula is still satisfiable, we mark the variable  $q$  as irrelevant. If  $S$  is unsatisfiable, we add the variable again to  $S$ . We continue this process with another  $Q$  variables until we have tried every  $Q$  variable. All  $Q$  variables that are marked as irrelevant for satisfiability of  $S$  are removed from the clause that is appended to  $R$ . The irrelevant variables are added to  $S$  again as they are only irrelevant in respect to the current assignment to  $Q$ .

## 4.2 Learning to Learn Clauses

The heuristic that shortens clauses requires to solve a SAT instance for each  $Q$  variable. We want to reduce that effort. Consider the previous example where  $Q$  contains the variables  $q_1, q_2, q_3$ , and  $q_4$ . In hindsight, we know that, if we remove  $q_2$  and  $q_4$  from  $S$ , then  $S$  is still satisfiable under the assignment  $q_1 = \text{true}$ ,  $q_3 = \text{false}$ . Thus, a single SAT instance is sufficient to establish that  $q_2$  and  $q_4$  can be removed from the clause to be added to  $R$ . Of course, we do not know in advance which variables can be successfully removed.

However considering practical applications, we suspect that in most iterations the same variables are excluded from the learned clauses. In a diagnostic expert system, for example, the  $Q$  variables represent symptoms. We want to determine if there are possibly symptoms that can prove a particular defect. Typically, only a subset of the symptoms can be linked to the defect. The  $Q$  variables that are irrelevant for the defect correspond to the variables that are irrelevant to satisfy formula  $S$ . Experimental results have confirmed our observation. See Section 5.

Instead of removing  $Q$  variables one-by-one and solving each time a SAT instance, we remove groups of variables that are likely irrelevant. We estimate the likelihood of irrelevance of a variable by computing the percentage of learned clauses from which the variable was previously removed. The variables with the highest estimate are removed tentatively within one step, and then  $S$  is solved. If  $S$  is unsatisfiable, the algorithm tries to remove the variables one-by-one. The remaining  $Q$  variables are always processed one-by-one. Observe that, in randomly generated instances that have no structure, we may not be able to predict sets of irrelevant variables.

## 5. COMPUTATIONAL RESULTS

In order to implement the solution algorithm, we have selected the SAT solver zChaff by Zhang, Madigan, Moskewicz, and Malik [15]. The solver zChaff has proven to perform very well on the annual SAT Competitions [12] conducted during the International Conference on Theory and Applications of Satisfiability Testing. The implementation includes learning and shortening of clauses. We call the resulting solver QallSAT. A preliminary version of the learning scheme that guides the selection of variable groups to be removed as described in Section 4.2 has been implemented by a second version of the Q-ALL SAT solver. We used two sets of benchmark problems by Remshagen and Truemper [8] that represent a robot navigation problem and a planning problem in a game. We have implemented and tested the solution algorithm on a UltraSPARC-II (400 MHz).

**Table 1: Test results for robot instances**

Instance	QallSAT (sec)	QRSsat (sec)	QuBERel1.3 (sec)
robot10no1	16.7	<0.1	201.3
robot10no2	13.6	0.9	92.9
robot10no3	10.7	0.9	499.0
robot10yes1	35.8	7.5	252.7
robot10yes2	6.3	1.9	250.2
robot10yes3	1.9	3.8	506.3

The first set of benchmarks contains 6 instances. Each instance has 100  $Q$  variables, 100  $X$  variables, and 200  $Y$  variables. There are 1681–1683 clauses in  $R$  and 1123–1125 clauses in  $S$ . The first three robot instances are *true*. The remaining three instances are *false*. When we disabled temporarily learning of short clauses, no instance could be solved within 60 min. With learning of clauses, the six instances were solved in 1.9 sec to 35.8 sec. The second column in Table 1 show the computational results of QallSAT.

We used 14 instances from the second set of benchmarks. Based on experimental results with different Q-ALL SAT and QBF solvers, the selected instances contain four instances that are considered to be very hard. The last four rows in Table 2 refer to the hard game instances. The first 10 instances have 15  $Q$  variables. The remaining four hard instances have 25  $Q$  variables. Each game instance has 275–405  $Y$  variables and no  $X$  variables. Formula  $R$  contains only one clause. Formula  $S$  contains 781–841 clauses. The first six and last four instances evaluate to *true*. The other four instances are *false*. Without learning of short clauses, even instances that are considered to be easy have a run-time of at least 20 min. With learning of clauses, the four difficult instances have a run-time of 3.2–24.1 sec. The remaining eight instances are solved within 1.3 sec, except for game\_15\_5 which has a run-time of 77 sec. See the second column in Table 2 for detailed results on the game problems.

**Table 2: Test results for game instances**

Instance	QallSAT (sec)	QRSsat (sec)	QuBERel1.3 (sec)
game_15_0	0.8	4.7	2.3
game_15_1	0.8	2.8	14.4
game_15_2	1.3	4.7	4.3
game_15_3	0.8	0.9	30.2
game_15_4	8.0	1.9	16.9
game_15_5	77.0	22.5	67.0
game_15_6	0.9	12.9	0.1
game_15_7	0.4	0.9	0.1
game_15_8	0.4	<0.1	0.1
game_15_9	0.4	0.9	0.1
game_25_0	6.1	390.6	>3600
game_25_1	9.5	2998.6	>3600
game_25_2	24.1	246.0	>3600
game_25_3	3.2	267.5	>3600

We also tested whether the run-times improve when several  $Q$  variables were removed in one step to shorten the learned clauses as described in Section 4.2. With the current implementation the run-times are cut into half on average. Analysis of the test instances suggest that the run-times can be improved further if the size of the variable groups is adjusted appropriately for each benchmark. In case of the

game instances, the same set of about 3–6 variables are typically contained in the clauses after shortening that clause. In case of the robot instances, almost all but one variable was removed from the clauses in each case. The current selection process often chooses too many  $Q$  variables that cannot be removed together. Thus, the variables in that group have still to be processed one-by-one.

We compare the run-times with the performance of the Q-ALL SAT solver QRSsat by Remshagen and Truemper [8] which is based on backtracking-search. The third column of Table 1 and 2 shows the run-times of QRSsat. On the robot instances QRSsat, performs better than QallSAT. In particular, QRSsat performs much better on the first three robot instances which evaluate to *true*. However, on the robot instances QallSAT is faster, except for game.15.5. On average, QallSAT is by a factor of about 30 faster than QRSsat on the game instances. QallSAT performs particularly well on the four hard game instances compared with QRSsat.

We also transformed each Q-ALL SAT instance into an equivalent 2QBF instance in order to compare the performance of QallSAT with a QBF solver. The transformation of the robot instances to QBF increases the number of variables significantly as the transformation introduces a new variable for each clause in  $R$ . Since each game instance has only one clause in  $R$ , the game instances increase only slightly. We have selected the QBF solver QuBERel1.3 by Giunchiglia, Narrizano, and Tacchella [3], [4]. QuBERel1.3 has performed very well on the selected benchmarks compared with other QBF solvers [8]. The fourth column of Table 1 and 2 displays the run-times of the QBF solver QuBERel1.3. QallSAT is on average by a factor of about 20 faster than QuBERel1.3 on the robot instances. On the first ten game instances, QuBERel1.3 performs quite well. The average run-time of QuBERel1.3 on these instances is 13.5 sec. QallSAT has an average run-time of 9.1 sec on these instances. The QBF solver could not solve any of the four game instances that are considered to be very hard within 60 min.

## 6. CONCLUSION

The Q-ALL SAT solver proposed in this paper reduces Q-ALL SAT to a sequence of satisfiability problems. Heuristics to learn short clauses are applied to reduce the search space. Our solver QallSAT has been tested on two sets of benchmark problems. Compared with the backtracking-search based Q-ALL SAT solver QRSsat, QallSAT is on average about 30 times faster on one of the benchmarks. Compared with the QBF solver QuBERel1.3, QallSAT is on average 20 times faster on the first benchmarks and over 100 times faster on the second benchmarks.

A direct attack of Q-ALL SAT seems to be more successful than a general QBF solver on the transformed instances. Our solution approach is very promising though it does not always outperform the Q-ALL SAT solver QRSsat. However, it exposes more uniform run-times that never exceed 2 min on any given test instance. Further work should be conducted to improve the heuristic that guesses groups of irrelevant variables.

## 7. REFERENCES

- [1] 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.
- [2] T. Eiter and G. Gottlob. The Complexity of Logic-based Abduction. *Journal of the Association for Computing Machinery* 42:3–42, 1995.
- [3] E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for Quantified Boolean Logic Satisfiability. In *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002.
- [4] E. Giunchiglia, M. Narrizano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artificial Intelligence*, 145(1):99–120, 2003.
- [5] M. Mneimneh and K. Sakallah. Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution. *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing*, 2003.
- [6] C. Papadimitriou. Computational Complexity. *Addison-Wesley*, 1994.
- [7] D. Ranjan, D. Tang, and S. Malik. A comparative study of 2qbf algorithms. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.
- [8] A. Remshagen and K. Truemper. An Effective Algorithm for the Futile Questioning Problem. *Journal of Automated Reasoning*, 34(1):31–47, 2005
- [9] J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
- [10] J. Rintanen. Partial implicit unfolding in the davis-putnam procedure for quantified boolean formula. In *International Conference on Logic Programming, Artificial Intelligence and Reasoning*, pages 362–376, 2001.
- [11] SAT 2005. Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing. Fahiem Bacchus and Toby Walsh (Eds.), Springer Verlag, *Lecture Notes in Computer Science 3569*, 2005.
- [12] SAT Competitions. <http://www.satcompetition.org>. Daniel Le Berre and Laurent Simon (Organizing committee). 2002–2005.
- [13] K. Truemper. Design of Logic-based Intelligent Systems. *Wiley*, 2004.
- [14] H. Zhang. SATO: an Efficient Propositional Prover. In *Proceedings of the International Conference on Automated Deduction (CADE-97)*, pages 272–275, 1997.
- [15] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design*, 2001.
- [16] L. Zhang and S. Malik. 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*, 2002.
- [17] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design*, 2002.