

# An Effective Algorithm for the Futile Questioning Problem

Anja Remshagen and Klaus Truemper

## Abstract

In the futile questioning problem, one must decide whether acquisition of additional information can possibly lead to the proof of a conclusion. Solution of that problem demands evaluation of a quantified Boolean formula at the second level of the polynomial hierarchy. The same evaluation problem, called Q-ALL SAT, arises in many other applications. In this paper, we introduce a special subclass of Q-ALL SAT that is at the first level of the polynomial hierarchy. We develop a solution algorithm for the general case that utilizes backtracking search and a new form of learning of clauses. Results are reported for two sets of instances involving a robot route problem and a game problem. For these instances, the algorithm is substantially faster than state-of-the-art solvers for quantified Boolean formulas.

## 1 Introduction

An important application of quantified Boolean formulas arises from questioning in expert systems. In the typical setting, the expert system repeatedly queries the user for additional information until the desired conclusion can be proven. For example, in an expert system for medical diagnosis, the expert system asks for symptom values to prove presence of a disease  $d$ . Suppose that disease  $d$  is not present. Then this will be established only when all possible symptom questions have been asked and answered. Yet, one may be able to draw the same conclusion much earlier, by proving that, no matter which of the so-far-unexplored symptoms are present, one will be unable to establish disease  $d$ . If this is so, we say that further questioning about symptoms is *futile*.

One establishes whether the situation of futile questioning is at hand as follows. Let  $Q$  be the set of symptom variables. Define  $R$  to be the conjunctive normal form (CNF) formula describing the relationships among the  $Q$  variables. Thus, an assignment to the symptom variables  $Q$  can be extended to a satisfying solution of  $R$  if and only if that assignment represents feasible symptoms. We call such an assignment *R-acceptable*. Let  $S$  be a CNF formula describing the relationships between symptoms and diseases. We assume that  $S$  contains the clause with the single literal  $\neg d$ . Hence, all satisfying solutions of  $S$  enforce the value *False* for disease  $d$ . An assignment to  $Q$  that cannot be extended to a satisfying solution of  $S$ , is called *S-unacceptable*. Since any satisfying solution of  $S$  must have  $d = \text{False}$ , the case of an *S-unacceptable* assignment for  $Q$  is at hand if and only if presence of the disease  $d$  can be proven using that assignment.

By these definitions, presence of  $d$  can possibly be proven if and only if there exists an  $R$ -acceptable and  $S$ -unacceptable assignment to  $Q$ . Thus, the opposite case of futile questioning is at hand if no such assignment exists.

We express the problem of deciding futility of questioning by a quantified Boolean formula. Assume that the variable sets of  $R$  and  $S$  are  $Q \cup X$  and  $Q \cup Y$ , respectively, where  $Q$ ,  $X$ , and  $Y$  are pairwise disjoint. Then the case of futile questioning is at hand if and only if the quantified Boolean formula

$$\exists Q(\exists X R \wedge \forall Y \neg S) \tag{1}$$

evaluates to *False*. We refer to the problem of evaluating the formula (1) as Q-ALL SAT. The existence of a polynomial-time solution algorithm is unlikely, since Q-ALL SAT is  $\Sigma_2^P$ -complete. Thus, the problem is at the second level of the polynomial hierarchy. For the polynomial hierarchy, see Stockmeyer [1976].

The problem Q-ALL SAT arises not only in medical diagnosis, but also in other applications, like, for example, in conformant planning and in regulatory compliance (Straach and Truemper [1999]). Further applications are discussed in Eiter and Gottlob [1995]. In case of conformant planning, one wants to determine a valid plan that leads to the goal regardless of uncertain conditions. In this paper we consider a planning problem for robot navigation and for a game problem.

Current algorithms for deciding the value of quantified Boolean formulas cannot evaluate formula (1) directly, since they require as input a formula where all quantifiers precede a single CNF formula. Let QBF denote the problem of deciding the value of formulas in the latter format. Though a direct solution of Q-ALL SAT by QBF solvers is not possible, one may transform an instance of Q-ALL SAT to an instance of QBF and then use any QBF solver.

The transformation of an instance of Q-ALL SAT to an instance of QBF can be done compactly via introduction of one additional variable for each clause in  $R$  plus one additional variable which represents whether  $R$  is satisfied. In practical applications, the number of clauses in  $R$  exceeds by far the total number of variables. For example, in one of our test instances, *robot\_8\_1*, the number of variables of the Q-ALL SAT instance is 256, while the number of clauses of  $R$  is 925. The resulting number of variables is  $256 + 925 + 1 = 1182$  and thus is almost five times the original number of variables. The total number of clauses increases significantly as well. In the case of *robot\_8\_1*, the total number of clauses rises from 1521 to 3211. The increase in size is only linear, but the effect is nevertheless undesirable. Another disadvantage of the transformation is that it tends to hide structural properties of Q-ALL SAT that one may be able to exploit in a direct attack on that problem.

In this paper, we propose a solution algorithm for Q-ALL SAT that deals with the subformulas  $R$  and  $S$  directly, see Section 3. Our approach is based on backtracking search and uses a new form of learning clauses. We also define a special subclass of Q-ALL SAT called 2CUT Q-ALL SAT that is motivated by a planning problem. Each instance of the subclass can be solved via a polynomial number of instances of the satisfiability problem SAT of propositional logic.

Section 4 introduces the subclass. Section 5 describes the sets of Q-ALL SAT instances. The last section of the paper presents computational results of our solution method on two classes of test instances and compares them with the performance of state-of-art QBF solvers on QBF instances derived from the test instances by suitable transformations. The results show that, for these two classes, the direct attack on Q-ALL SAT is quite effective and significantly faster than application of the QBF solvers to the QBF instances produced by the transformations.

## 2 Prior Work

Recent work on QBF solvers is as follows. Kleine Büning, Karpinski, and Flögel [1995] propose a complete and sound resolution-based algorithm. However, due to the possibly exponential growth of the formula in the resolution process, these algorithms are not practical. Among the most promising algorithms for QBF are backtracking search algorithms, which have been proven to be very successful in solving the satisfiability problem SAT (Bayardo and Schrag [1997], João, Marques-Silva, and Sakallah [1999], Zhang [1997], Zhang, Madigan, Moskewicz, and Malik [2001]). Cadoli, Schaerf, Giovanardi, and Giovanardi [2003] 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 [2002], [2003] implement a learning scheme in their solver QuBE. Rintanen [2001] extends the Davis-Putnam procedure by unit propagation for quantified formulas. The QBF solver Quaffle by Zhang and Malik [2002] learns conflict clauses during the solution process. A satisfiability-driven learning scheme by Zhang and Malik [2002] augments the CNF formula by a formula in disjunctive normal form. The solver QSAT<sub>CNF</sub> by Plaisted, Biere, and Zhu [2003] is based on a DPLL procedure and uses the SAT solver SATO (Zhang [1997]). The solver QSAT<sub>CNF</sub> can be exponentially faster than BDDs on some examples.

The problem Q-ALL SAT is closely related to logic-based abduction. Eiter and Gottlob [1995] describe several applications and investigate the complexity of logic-based abduction. Conformant planning has been tackled by several approaches. For example, Rintanen [1999] shows that conformant planning is  $\Pi_2^P$ -hard and translates conditional planning problems to instances of QBF. Cimatti and Roveri [2000] suggest a representation based on Binary Decision Diagrams and develop a solution approach based on symbolic model checking. Bonet and Geffner [2000] use heuristic search algorithms to solve planning problems. Castellini, Giunchiglia, and Tacchella [2003] apply a SAT-based approach to systems specified in the action language  $\mathcal{C}$ .

### 3 The Solution Algorithm

In this section, we describe a solution algorithm for Q-ALL SAT called QRSsat. It uses a backtracking scheme while it searches for an  $R$ -acceptable and  $S$ -unacceptable assignment to  $Q$ . Backtracking has been shown to be effective in SAT algorithms when combined with dynamic learning of clauses. For example, see the solver relsat by Bayardo and Schrag [1997], GRASP by João, Marques-Silva, and Sakallah [1999], the SATO by Zhang [1997], or zChaff by Zhang, Madigan, Moskewicz, and Malik [2001].

We call an  $R$ -acceptable and  $S$ -unacceptable assignment to the  $Q$  variables of a Q-ALL SAT instance a *solution*. In QRSsat, two situations cannot lead to a solution of a Q-ALL SAT instance and thus trigger backtracking in the search tree:

1. A truth assignment for some  $Q$  variables produces an empty clause in  $R$ .
2. A truth assignment for some  $Q$  variables leads to satisfiability in  $S$ , no matter to which truth values the remaining  $Q$  variables are set.

In the first case, the reason for the empty clause can be determined from the search tree and can be added as an  $R$ -conflict clause to  $R$ . Any learning scheme used in SAT solvers, for example, any one of the dynamic learning schemes of the references cited above for the SAT problem can be applied in that step.

The second case occurs if the current assignment for some variables in  $Q$  satisfies all clauses of  $S$ . In that case, we define from the truth assignment of the fixed  $Q$  variables a clause that contains each fixed  $Q$  variable and that cuts off that assignment in the search tree. For example, assume that (1) the given Q-ALL SAT instance contains the three  $Q$  variables  $q_1, q_2, q_3$ , (2) the search tree sets  $q_1$  to *True* and  $q_2$  to *False*, and (3) the formula  $S$  is satisfiable under that assignment. Then the assignment for  $q_1$  and  $q_2$  cannot be extended to a solution of the Q-ALL SAT instance, and hence every solution must set  $q_1$  to *False* or  $q_2$  to *True*. In other words, a solution must satisfy the clause  $\neg q_1 \vee q_2$ . Let us call such a clause an  $S$ -conflict clause. The clause could be added to  $R$ , but would be useless for the tree search since it is always satisfied after the succeeding backtracking step and thus does not reduce the search space. Accordingly, we first sharpen each  $S$ -conflict clause by the following heuristic.

First, all free  $Q$  variables are removed from the clauses of  $S$ . If the formula  $S'$  resulting from  $S$  is unsatisfiable, the heuristic cannot sharpen the  $S$ -conflict clause. Otherwise, each fixed  $Q$  variable  $q_i$  is processed as follows. The variable  $q_i$  is deleted from the current  $S'$ . If the reduced formula is unsatisfiable, the variable  $q_i$  is added back again. Otherwise,  $S'$  denotes the reduced formula, and the literal of  $q_i$  in the  $S$ -conflict clause is removed from that clause.

If the processing of the fixed  $Q$ -variables removes at least one literal from the original  $S$ -conflict clause, then the final, sharpened  $S$ -conflict clause is added to  $R$ .

We carry out sharpening the process for an example where  $q_1 = \textit{True}$  and  $q_2 = \textit{False}$  have produced the  $S$ -conflict clause  $\neg q_1 \vee q_2$ . Let  $S$  be the CNF

formula with the following clauses.

$$\begin{aligned} &\neg q_1 \vee q_2 \vee \neg y_1 \\ &q_1 \vee \neg y_2 \\ &q_1 \vee \neg q_2 \vee y_2 \end{aligned}$$

Observe that both of the two  $Q$  variables are fixed. Thus, the formula has no free  $Q$  variables that have to be removed. We try to sharpen the  $S$ -conflict clause by first removing  $q_2$  tentatively from all clauses. We obtain the clauses

$$\begin{aligned} &\neg q_1 \vee \neg y_1 \\ &q_1 \vee \neg y_2 \\ &q_1 \vee y_2 \end{aligned}$$

Since the reduced formula is satisfiable under  $q_1 = \text{True}$ , we remove the literal  $q_2$  from the  $S$ -conflict clause, thus getting the unit clause  $\neg q_1$ . In the next step, we tentatively remove  $q_1$ . The resulting formula, consisting of the three unit clauses  $y_1$ ,  $\neg y_2$ , and  $y_2$ , is unsatisfiable. Since we have processed all  $Q$  variables that have been fixed in the tree, the  $S$ -conflict clause  $\neg q_1$  cannot be further sharpened, and we add that clause to  $R$ .

The heuristic for sharpening  $S$ -conflict clauses deletes variables first that have been fixed last in the search tree. Thus, the variables fixed most recently are more likely to be removed than the variables fixed in the beginning of the search process. Accordingly, the new clauses will more likely lead to nonlinear backjumping, that is, the algorithm backtracks more than one node in the search tree.

The heuristic for sharpening  $S$ -conflict clauses requires the solution of several satisfiability problems arising from  $S$ , and thus may seem to be a considerable computational burden if the SAT instance  $S$  is hard. But the SAT instances that must be solved are derived from  $S$  by deleting or fixing  $Q$  variables, and the resulting CNF formulas tend to be rather easy even if the original  $S$  instance is hard.

We summarize the algorithm.

*Input:*  $R$  with variable set  $Q$  and  $X$ .  $S$  with variable set  $Q$  and  $Y$ .  
 $X$  and  $Y$  are disjoint.

*Output:* A solution, or “there is no solution.”

1. Select the next  $Q$  variable and a truth value. If all variables have been tried, output “there is no solution,” and stop. Otherwise, fix the selected variable to the chosen value in both  $R$  and  $S$ , and extend the search tree accordingly.
2. Do unit-resolution in  $R$  on all variables and in  $S$  on the variables  $Y$ .
3. If the fixed  $Q$  variables produce an empty clause in  $S$  or if all  $Q$  variables are fixed:

- (a) Solve  $R$ .
  - (b) If  $R$  is unsatisfiable, find an  $R$ -conflict clause, and add it to  $R$ . Backtrack in the search tree, and go to Step 1.
  - (c) ( $R$  is satisfiable and all  $Q$  variables are fixed.) Solve  $S$ . If  $S$  is unsatisfiable, output the values of the  $Q$  variables, and stop. Otherwise, derive the  $S$ -conflict clause, attempt to sharpen it, and, if successful, add the sharpened version to  $R$ . Backtrack in the search tree, and go to Step 1.
4. If the fixed  $Q$  variables produce an empty clause in  $R$  and thus make  $R$  unsatisfiable, find an  $R$ -conflict clause, and add it to  $R$ . Backtrack in the search tree, and go to Step 1.
  5. If the fixed  $Q$  variables satisfy all clauses of  $S$ , derive the  $S$ -conflict clause, attempt to sharpen it, and, if successful, add the sharpened version to  $R$ . Backtrack in the search tree, and go to Step 1.
  6. Go to Step 1.

In Step 1, the next variable is selected by a version of the MOMS (Maximum Occurrences in Minimum Size clauses) heuristic adapted to Q-ALL SAT as follows. For each  $Q$  variable  $q$ , we count the nonnegated occurrences of  $q$  in  $R$  and the negated occurrences in  $S$  in all clauses of length  $i$ . Let this number be  $p_q(i)$ . Analogously, let  $n_q(i)$  be the number of negated occurrences of  $q$  in all clauses of  $R$  of length  $i$  and nonnegated occurrences of  $q$  in  $S$  in clauses of length  $i$ . Define a vector  $h_q$  whose  $i$ th element is

$$h_q(i) = p_q(i) + n_q(i) - \frac{1}{3} \cdot |p_q(i) - n_q(i)| \quad (2)$$

We select the variable  $q$  with the lexicographically largest vector  $h_q$ . Thus, variables are first compared with respect to  $h_q(1)$ . The selected variable  $q$  is set to *True* if the sum of all  $p_q(i)$  is larger than the sum of all  $n_q(i)$ ; otherwise, it is set to *False*. The selected variable and truth value aim at inducing satisfiability in  $R$  and unsatisfiability in  $S$ .

Observe, that in Step 2 unit resolution cannot be applied in formula  $S$  to the  $Q$  variables since we need to find an assignment for  $Q$  that results in unsatisfiability of  $S$ .

## 4 The Class 2CUT Q-ALL SAT

The problem Q-ALL SAT is at the second level of the polynomial hierarchy and thus very hard. However, practical applications may produce a particular structure that allows comparatively rapid solution. One such class arises from certain problems in conformant planning. In the general case of the planning problem, one must find a valid and successful plan or one must determine that such a plan does not exist. In the corresponding instance of Q-ALL SAT, the

$Q$  variables represent the different actions that can be taken, while some of the  $Y$  variables of  $S$  represent possible failures. A plan is *valid* if it corresponds to an  $R$ -acceptable assignment to the  $Q$  variables. Formula  $S$  has a clause that enforces at least one failure to occur. A plan is *successful* if it corresponds to an  $S$ -unacceptable assignment to  $Q$ . Hence, a valid and successful plan corresponds to an  $R$ -acceptable and  $S$ -unacceptable assignment to  $Q$ .

An example is a planning problem for robot navigation where one either determines a possible route of a robot that leads to the destination no matter how obstacles might be placed according to some given rules, or one concludes that such a route does not exist. Each  $Q$  variable represents a possible move of the robot. A valid plan is an  $R$ -acceptable assignment to the  $Q$  variables. A plan is unsuccessful if one of the selected moves of the robot leads to a collision with an obstacle. The collisions with possible obstacles are represented by some  $Y$  variables. Each collision implies that a particular move was selected. Thus, formula  $S$  contains clauses equivalent to implications of the form  $y \Rightarrow q$  with  $y \in Y$  and  $q \in Q$ . In addition, formula  $S$  contains conditions for the possible positions of the obstacles and a clause that enforces at least one collision to be *True*. Hence, all satisfiable solutions of  $S$  represent unsuccessful plans.

We discuss a small example. The variable sets of  $S$  are  $Q = \{q_1, q_2\}$  and  $Y = \{y_1, y_2, z_1, z_2\}$  with the following interpretation. The variables  $q_1$  and  $q_2$  represent two actions. Each of the two variables  $y_1$  and  $y_2$  represents a collision or a possible failure of a plan, which can only occur if the action  $q_1$  or  $q_2$ , respectively, is taken. The implication  $y_i \Rightarrow q_i$ , for  $i = 1, 2$ , enforces that condition. Each of the variables  $z_i$ ,  $i = 1, 2$ , is an additional conclusion implied by the failure  $y_i$ . Thus,  $S$  contains CNF clauses equivalent to  $y_i \Rightarrow z_i$ , for  $i = 1, 2$ . Finally,  $S$  contains the clause  $y_1 \vee y_2$ , which enforces that all satisfying solutions for  $S$  represent unsuccessful plans. We summarize the CNF formula  $S$ :

$$\begin{aligned}
S = & (q_1 \vee \neg y_1) \wedge & (3) \\
& (q_2 \vee \neg y_2) \wedge \\
& (\neg y_1 \vee z_1) \wedge \\
& (\neg y_2 \vee z_2) \wedge \\
& (y_1 \vee y_2)
\end{aligned}$$

In the general case, the planning problem has the following structure.

1. The variable set  $Y$  can be partitioned into two sets  $Y_1$  and  $Y_2$  so that the following holds.
  - (a) All variables in  $Y_1$  occur only negated in  $S$ , except for one special clause whose literals are the nonnegated variables of  $Y_1$ . In the example (3) of  $S$ ,  $Y$  is partitioned into  $Y_1 = \{y_1, y_2\}$  and  $Y_2 = \{z_1, z_2\}$ , and the special clause is  $y_1 \vee y_2$ .
  - (b) All clauses that contain at least one  $Q$  variable have at least one  $Y_1$  variable as well.

- (c) For each variable  $y$  in  $Y_1$ , the following property holds. Consider all clauses that contain the literal  $\neg y$  and at least one  $Q$  variable. Let  $Q(y)$  be the set of  $Q$  variables occurring in these clauses. In each of these clauses, at most one of the variables in  $Q(y)$  does not occur. In the example (3) of  $S$ , we have  $Y_1 = \{y_1, y_2\}$ , and for each  $y_i \in Y_1$ , the clause  $q_i \vee \neg y_i$  contains  $y_i$  and  $Q$  variables. Thus  $Q(y_i) = \{q_i\}$ , for  $i = 1, 2$ , and the condition is trivially satisfied.

2. All variables in  $Q$  occur only nonnegated in  $S$ .

We call the class of Q-ALL SAT instances with this structure 2CUT Q-ALL SAT. Note that the above conditions only constrain the CNF formula  $S$  and do not impose any restrictions on  $R$ . Hence, 2CUT Q-ALL SAT is still NP-hard.

In a typical practical application of 2CUT Q-ALL SAT, the variable set  $Y_1$  represents the set of all possible failures. Each failure  $y \in Y_1$  occurs only if a specific action  $q \in Q$  is taken, resulting in the clause  $\neg y \vee q$ . The variables  $Q$  do not occur in other clauses of  $S$ . The special clause of property 1(a) enforces at least one failure. If all  $Y_1$  variables occur negated in the remaining clauses of  $S$ , all conditions of the properties 1 and 2 are met.

We show that, due to the structure of 2CUT Q-ALL SAT, one can learn constraints, also called *cuts*, which eliminate solutions and which can be expressed as CNF clauses with at most two literals. Before proving this fact for the general case, we illustrate that situation using the example (3).

Assume that a truth assignment to  $\{q_1, q_2\}$  is a valid but unsuccessful plan. Thus, the assignment can be extended to a satisfying solution  $\alpha$  in  $S$ . Due to the clause  $y_1 \vee y_2$ , at least one of the variables  $y_i$  is set to *True* by  $\alpha$ , say  $y_1$ . Then  $q_1$  has to be *True* in order to satisfy the clause  $q_1 \vee \neg y_1$ . No matter which values the remaining  $Q$  variables have, the CNF formula  $S$  evaluates to *True* if we set all variables in  $Y_1$  except for  $y_1$  to *False*, and if we set the  $Y_2$  variables to the truth value given by  $\alpha$ . Hence, any assignment to the  $Q$  variables that sets  $q_1$  to *True* is  $S$ -acceptable and not a solution of the Q-ALL SAT instance. Thus, all solutions of the Q-ALL SAT instance must satisfy the clause  $\neg q_1$ .

Now consider the general case of a 2CUT Q-ALL SAT instance with CNF formulas  $R$  and  $S$ . Assume an  $R$ -acceptable assignment to  $Q$  is  $S$ -acceptable and thus can be extended to a satisfying solution  $\alpha$  of  $S$ . Then the special clause of property 1(a), which contains all variables of  $Y_1$  nonnegated, forces at least one variable  $y$  in  $Y_1$  to have the value *True*. Indeed, by property 1(a), all variables of  $Y_1$  occur always negated in  $S$  except for the special clause, and thus we may suppose that  $y$  is the only variable with assigned value *True*. Due to property 1(a) and 1(b), a clause with a  $Q$  variable and without literal  $\neg y$  contains at least one negated variable of  $Y_1 - \{y\}$ . That clause evaluates to *True* under the values assigned to  $Y_1$ . Since  $\alpha$  is a satisfying solution of  $S$ , all clauses without  $Q$  variables evaluate to *True* under the values assigned to  $Y_1$  and  $Y_2$  as well. Accordingly, all clauses that do not contain the literal  $\neg y$  and a variable in  $Q$  evaluate to *True* when just the values assigned to the variables

of  $Y_1$  and  $Y_2$  are used. We show that the remaining clauses are satisfied by the truth assignment to  $Y_1$  and  $Y_2$  and by the values of at most two  $Q$  variables.

Suppose that at least two variables in  $Q(y)$  have value *True* assigned, say,  $q_1 = \text{True}$  and  $q_2 = \text{True}$ . Every clause with literal  $\neg y$  and with at least one  $Q$  variable contains literal  $q_1$  or  $q_2$  by definition of  $Q(y)$  and by property 1(c). Thus, these clauses evaluate to *True* under the assignment  $q_1 = \text{True}$  and  $q_2 = \text{True}$ . Therefore and due to the above conclusion, any assignment to  $Q$  with  $q_1 = \text{True}$  and  $q_2 = \text{True}$  is  $S$ -acceptable. We may add the clause  $\neg q_1 \vee \neg q_2$  to  $R$  to rule out all such assignments.

Suppose that just one variable in  $Q(y)$  has value *True* assigned, say,  $q_1 = \text{True}$ . Arguing as in the above case, any assignment to  $Q$  with  $q_1 = \text{True}$  is  $S$ -acceptable, and we may add the clause  $\neg q_1$  to  $R$  to rule out all such assignments. Finally, suppose that all variables of  $Q(y)$  have the value *False* assigned. Since  $Q$  variables occur only nonnegated in the clauses of  $S$  due to property 2, there is no way to choose an assignment to  $Q$  that is  $S$ -unacceptable.

We have shown that for every  $S$ -acceptable assignment to  $Q$ , either we can add a clause to  $R$  with at most two literals that rules out that assignment, or we conclude that there exists no  $S$ -unacceptable assignment to  $Q$ . Observe that the solution algorithm proposed in Section 3 can sharpen every  $S$ -conflict clause of an instance of 2CUT Q-ALL SAT to contain at most two literals in linear time. Since the number of learned clauses is at most quadratic in the number of  $Q$  variables, the solution algorithm terminates after solving a polynomial number of instances of the satisfiability problem SAT. This result implies that the class 2CUT Q-ALL SAT is at the first level of the polynomial hierarchy. Also, from properties 1 and 2 it is evident that the instances of 2CUT Q-ALL SAT can be identified in polynomial time.

## 5 The Test Problems

We designed two sets of benchmark problems that, depending on choice of parameters, can be made progressively more difficult. At the same time, the problems are structured and, by their design, are closer to real-world problems than randomly generated instances. The first problem set is a navigation problem where a robot has to navigate through a grid with obstacles. The possible positions of obstacles are constrained. The instances differ by the grid size and by randomly placing some obstacles on the grid. The problem asks one whether the robot can reach its destination no matter how the obstacles are placed according to the constraints. A satisfying solution of the first formula  $R$  models a path of the robot for one possible constellation of the obstacles. The path is represented by the truth assignment for the  $Q$  variables. The assignment for the  $Q$  variables is  $S$ -unacceptable if it is impossible to place obstacles that interfere with the corresponding path of the robot.

Table 1: Robot navigation instances

| instance   | solution | $ Q $ | $ X $ | $ Y $ | # clauses<br>in $R$ | # clauses<br>in $S$ |
|------------|----------|-------|-------|-------|---------------------|---------------------|
| robot_8_1  | yes      | 64    | 64    | 128   | 925                 | 596                 |
| robot_8_2  | yes      | 64    | 64    | 128   | 926                 | 597                 |
| robot_8_3  | yes      | 64    | 64    | 128   | 925                 | 596                 |
| robot_8_4  | no       | 64    | 64    | 128   | 925                 | 596                 |
| robot_8_5  | no       | 64    | 64    | 128   | 926                 | 597                 |
| robot_8_6  | no       | 64    | 64    | 128   | 924                 | 595                 |
| robot_9_1  | yes      | 81    | 81    | 162   | 1268                | 832                 |
| robot_9_2  | yes      | 81    | 81    | 162   | 1269                | 833                 |
| robot_9_3  | yes      | 81    | 81    | 162   | 1268                | 832                 |
| robot_9_4  | no       | 81    | 81    | 162   | 1268                | 832                 |
| robot_9_5  | no       | 81    | 81    | 162   | 1269                | 833                 |
| robot_9_6  | no       | 81    | 81    | 162   | 1267                | 831                 |
| robot_10_1 | yes      | 100   | 100   | 200   | 1682                | 1124                |
| robot_10_2 | yes      | 100   | 100   | 200   | 1683                | 1125                |
| robot_10_3 | yes      | 100   | 100   | 200   | 1682                | 1124                |
| robot_10_4 | no       | 100   | 100   | 200   | 1682                | 1124                |
| robot_10_5 | no       | 100   | 100   | 200   | 1683                | 1125                |
| robot_10_6 | no       | 100   | 100   | 200   | 1681                | 1123                |

Table 1 summarizes the robot instances. The second column in Table 1 indicates if the instance has a solution. Half of the instances have a solution. The next three columns show the number of variables in the sets  $Q$ ,  $X$ , and  $Y$ . The last two columns list the number of clauses in  $R$  and  $S$ . All robot instances belong to the subclass 2CUT Q-ALL SAT.

The second problem set models the search for a successful strategy in a game. The first of two players needs to decide on a set of moves that prevent the opponent from reaching a goal. The  $Q$  variables represent the decisions of the first player. The valid moves of the first player are constrained by formula  $R$ , which by a single clause demands that the first player makes at least one move. Formula  $S$  models a game tree. For a given decision, that is, for a truth assignment to the  $Q$  variables, the opponent can reach the goal if the CNF formula  $S$  is satisfiable under the assignment for  $Q$ . Thus, the first player wants to determine an  $S$ -unacceptable and  $R$ -acceptable assignment to  $Q$ .

The test instances are divided into 12 problem sets, each of which contains 12 instances. Table 2 summarizes the problem sets. The second column shows the number of instances with a solution for each set. The next three columns display the number of  $Q$ ,  $X$ , and  $Y$  variables of each instance. The last two columns show the number of clauses in  $R$  and  $S$ , where for  $S$  a range is given that applies to the 12 instances of the applicable problem set.

Table 2: Sets of game instances

| problem set | solutions | $ Q $ | $ X $ | $ Y $ | # clauses<br>in $R$ | # clauses<br>in $S$ |
|-------------|-----------|-------|-------|-------|---------------------|---------------------|
| game_15_20  | 6         | 15    | 0     | 275   | 1                   | 781–811             |
| game_15_50  | 5         | 15    | 0     | 305   | 1                   | 781–811             |
| game_15_100 | 4         | 15    | 0     | 355   | 1                   | 781–811             |
| game_15_150 | 4         | 15    | 0     | 405   | 1                   | 781–811             |
| game_20_20  | 7         | 20    | 0     | 275   | 1                   | 786–826             |
| game_20_50  | 10        | 20    | 0     | 305   | 1                   | 786–826             |
| game_20_100 | 9         | 20    | 0     | 355   | 1                   | 786–826             |
| game_20_150 | 8         | 20    | 0     | 405   | 1                   | 786–826             |
| game_25_20  | 8         | 25    | 0     | 275   | 1                   | 791–841             |
| game_25_50  | 8         | 25    | 0     | 305   | 1                   | 791–841             |
| game_25_100 | 8         | 25    | 0     | 355   | 1                   | 791–841             |
| game_25_150 | 10        | 25    | 0     | 405   | 1                   | 791–841             |

In order to compare our computational results with other solvers, we converted our test instances into the Q-DIMACS format required by QBF solvers; see QBFLIB [2002] for details about that format. We have chosen a transformation that results in a minimum number of quantifier alterations. For complexity reasons, one quantifier alteration is required. We explain the transformation using a small example. Let

$$R = (\neg q_2 \vee \neg q_3 \vee x_1) \wedge (\neg x_1 \vee q_1) \quad (4)$$

and define

$$S = (\neg q_1 \vee \neg q_2 \vee y_1) \wedge (\neg q_1 \vee \neg q_3 \vee y_2) \wedge \neg y_1 \quad (5)$$

In order to have one quantifier alteration only in the final formula, it is convenient that we first negate formula (1), getting

$$\forall Q(\forall X \neg R \vee \exists Y S) \quad (6)$$

We rearrange formula (6) to

$$\forall Q \forall X \exists Y \neg R \vee S \quad (7)$$

It is easy to see that, except for trivial cases,  $\neg R \vee S$  is not a CNF formula. We convert  $\neg R \vee S$  to an equivalent CNF formula that for any assignment to  $Q$ ,

$X$ , and  $Y$  is satisfiable if and only if  $\neg R \vee S$  evaluates to *True*. For a compact conversion, we first introduce a new variable  $s$  and get the following formula equivalent to  $\neg R \vee S$ .

$$(s \vee \neg R) \wedge (\neg s \vee S) \quad (8)$$

The subformula  $\neg s \vee S$  of (8) can be converted to a CNF formula  $S'$  by adding the literal  $\neg s$  to each clause of  $S$ . For the example case of  $S$  of (5), we have

$$\begin{aligned} S' = & (\neg s \vee \neg q_1 \vee \neg q_2 \vee y_1) \wedge \\ & (\neg s \vee \neg q_1 \vee \neg q_3 \vee y_2) \wedge \\ & (\neg s \vee \neg y_1) \end{aligned} \quad (9)$$

Since  $R$  is a CNF formula, the subformula  $s \vee \neg R$  of (8) is in disjunctive normal form (DNF), say

$$s \vee \neg R = s \vee D_1 \vee D_2 \vee \dots \vee D_m \quad (10)$$

where each  $D_i$  is a conjunction of literals. The conversion of  $s \vee \neg R$  to an equivalent CNF formula introduces a new variable  $r_i$  for each  $D_i$  and defines the equivalent formula

$$\begin{aligned} & (s \vee r_1 \vee r_2 \vee \dots \vee r_m) \wedge \\ & \bigwedge_{i=1}^m (\neg r_i \vee D_i) \end{aligned} \quad (11)$$

Using the distributive law, each term  $\neg r_i \vee D_i$  of (11) is converted to CNF. The resulting  $R'$  is the desired equivalent CNF formula.

In the example case of  $R$  of (4), the two CNF clauses call for two variables  $r_1$  and  $r_2$ , and (11) is

$$\begin{aligned} & (s \vee r_1 \vee r_2) \wedge \\ & (\neg r_1 \vee (q_2 \wedge q_3 \wedge \neg x_1)) \wedge \\ & (\neg r_2 \vee (x_1 \wedge \neg q_1)) \end{aligned} \quad (12)$$

We use the distributive law to convert (12) to the following CNF formula  $R'$ .

$$\begin{aligned} R' = & (s \vee r_1 \vee r_2) \wedge \\ & (\neg r_1 \vee q_2) \wedge \\ & (\neg r_1 \vee q_3) \wedge \\ & (\neg r_1 \vee \neg x_1) \wedge \\ & (\neg r_2 \vee x_1) \wedge \\ & (\neg r_2 \vee \neg q_1) \end{aligned} \quad (13)$$

Let  $Z$  be the set of the new variables  $s, r_1, r_2, \dots, r_m$ . Then formula (1) evaluates to *True* if and only if

$$\forall Q \forall X \exists Y \exists Z R' \wedge S' \quad (14)$$

evaluates to *False*. Since  $R'$  and  $S'$  are CNF formulas, (14) is an instance of QBF. The transformation shows that a new variable for each clause in  $R$  plus one additional variable are added.

Due to the transformation, the number of variables and clauses increases substantially. For example, the Q-ALL SAT instance `robot_8_1` has according to Table 1 a total of  $|Q| + |X| + |Y| = 256$  variables, a total of 1521 clauses in  $R$  and  $S$ , and 925 clauses in  $R$ . The corresponding instance in the Q-DIMACS format has  $256 + 925 + 1 = 1182$  variables and 3211 clauses. For the game instances, the increase in size is not so significant since formula  $R$  consists of one clause only.

## 6 Computational Results

We have implemented a first version of QRSsat. The implementation includes the learning of  $S$ -conflict clauses, but as yet does not compute  $R$ -conflict clauses. For the tests, we used the two sets of benchmark problems described in Section 5. All computational results were obtained on a Sun ULTRA 5 (400 MHz) workstation. We selected three solvers: Quaffle by Zhang and Malik [2002], QuBE by Giunchiglia, Narrizano, and Tacchella [2002], [2003], and Semprop by Letz [2002]. In particular, we selected the solvers QuBE and Semprop since they performed very well at the SAT 2003 QBF Evaluation; see SAT [2003].

Table 3: Results for the robot instances

| instance   | QRSsat              |               | Quaffle       | QuBE          | Semprop       |
|------------|---------------------|---------------|---------------|---------------|---------------|
|            | # $S$ -<br>conflict | time<br>(sec) | time<br>(sec) | time<br>(sec) | time<br>(sec) |
| robot_8_1  | 0                   | 0             | >1200         | 26            | >1200         |
| robot_8_2  | 2                   | 0             | >1200         | 18            | >1200         |
| robot_8_3  | 0                   | 1             | >1200         | 49            | 2             |
| robot_8_4  | 6                   | 2             | >1200         | 41            | >1200         |
| robot_8_5  | 2                   | 0             | >1200         | 8             | >1200         |
| robot_8_6  | 2                   | 1             | >1200         | 38            | >1200         |
| robot_9_1  | 0                   | 0             | >1200         | 520           | >1200         |
| robot_9_2  | 1                   | 1             | >1200         | 122           | >1200         |
| robot_9_3  | 1                   | 1             | >1200         | 257           | 81            |
| robot_9_4  | 7                   | 3             | >1200         | 129           | >1200         |
| robot_9_5  | 2                   | 1             | >1200         | 45            | >1200         |
| robot_9_6  | 2                   | 2             | >1200         | 178           | >1200         |
| robot_10_1 | 0                   | 0             | >1200         | 252           | >1200         |
| robot_10_2 | 1                   | 1             | >1200         | 250           | >1200         |
| robot_10_3 | 0                   | 1             | >1200         | 506           | 7             |
| robot_10_4 | 8                   | 8             | >1200         | 201           | >1200         |
| robot_10_5 | 2                   | 2             | >1200         | 92            | >1200         |
| robot_10_6 | 2                   | 4             | >1200         | 499           | >1200         |

Table 3 lists the results for the robot instances. The second column shows the number of  $S$ -conflict clauses learned by QRSsat. The third column displays the run times of QRSsat in seconds. The run times for the three QBF solvers are given in columns four, five, and six. The solution process was aborted if the run time exceeded 20 min.

The following comments apply to the performance of QRSsat. All instances were solved rapidly, indeed all except one were solved within 4 sec. The exceptional case required 8 sec. Up to eight  $S$ -conflict clauses were learned for each instance. To evaluate the effect of the learned clauses, we temporarily disabled the corresponding subroutine. For the instance robot\_8\_5, the run time was about 9 min. In all other cases without solution, the run time exceeded 20 min. Thus, the learning of conflict clauses was essential.

The data for the three QBF solvers can be summarized as follows. Quaffle could not solve any instance within 20 min, while QuBE solved each instance within 8–499 sec. With three exceptions, Semprop could not solve any instance within 20 min; the exceptions required 2–81 sec. When compared with the fastest of the three QBF solver, which for the robot instances is QuBE, QRSsat is faster by a factor of 112.

Table 4 displays the results for the game problem sets. The columns are arranged analogously to Table 3. Due to the large number of test instances, we list the average run time for each of the 12 problem sets. After each column with the average run times, the next column displays how many instances could not be solved within 20 min. In the calculation of the average run times, only the instances that are solved within 20 min by the corresponding algorithm are considered.

Table 4: Results for the game sets

| problem set | QRSsat          |            |      | Quaffle    |      | QuBE       |       | Semprop    |      |
|-------------|-----------------|------------|------|------------|------|------------|-------|------------|------|
|             | # $S$ -conflict | time (sec) | > 20 | time (sec) | > 20 | time (sec) | > 20* | time (sec) | > 20 |
| game_15_20  | 8–704           | 3.8        | 0    | 15.8       | 0    | 13.2       | 0     | 23.0       | 0    |
| game_15_50  | 4–179           | 4.0        | 0    | 47.8       | 1    | 34.4       | 2     | 33.8       | 0    |
| game_15_100 | 1–21            | 1.8        | 0    | 116.7      | 0    | 70.1       | 0     | 30.3       | 0    |
| game_15_150 | 1–32            | 2.5        | 0    | 298.0      | 0    | 128.7      | 2     | 26.3       | 0    |
| game_20_20  | 4–10848         | 69.9       | 0    | 44.7       | 4    | 192.6      | 2     | 305.6      | 2    |
| game_20_50  | 2–394           | 14.1       | 0    | 157.0      | 0    | 62.5       | 1     | 189.1      | 1    |
| game_20_100 | 2–36            | 2.9        | 0    | 50.6       | 2    | 1.1        | 4     | 143.6      | 2    |
| game_20_150 | 1–12            | 3.1        | 0    | 0.0        | 4    | 103.6      | 4     | 185.7      | 0    |
| game_25_20  | 0–4608          | 91.4       | 1    | 143.9      | 4    | 0.7        | 5     | 0.1        | 5    |
| game_25_50  | 2–2288          | 36.8       | 0    | 70.4       | 2    | 52.1       | 2     | 136.3      | 4    |
| game_25_100 | 2–96            | 4.2        | 0    | 0.0        | 4    | 5.4        | 4     | 4.7        | 5    |
| game_25_150 | 1–9             | 3.1        | 0    | 0.0        | 2    | 264.0      | 3     | 30.1       | 2    |

\* The number of instances that were not solved within 20 min includes 9 cases in which QuBE terminated with an error.

QRSsat solved each instance within 612sec except for one case, where the time limit of 20 min was exceeded. That limit was exceeded by the solver Quaffle in 23 cases, by QuBE in 20 cases (9 cases resulted in an error), and by Semprop in 21 cases. Most of the unsolved instances had no solution. In general, all QBF solvers needed significantly more run time on instances without a solution. For example, four instances in problem set game\_25\_100 have no solution. All three QBF solvers exceeded 20 min on each of these four instances. Each of the remaining eight instances could be solved within 32 sec by all QBF solvers, except for one instance where Semprop exceeded the time limit. Quaffle solved the remaining instances even in less than 1 sec. Our procedure QRSsat solved each of the twelve instances within 19 sec. On average, it needed 5.3 sec for the four instances without solution and 3.6 sec for the eight instances with solution.

We also disabled the subroutine for learning  $S$ -conflict clauses for the game problems. For example, the average run time on the first problem set game\_15\_20 was 118.9 sec. The run time exceeded 200 sec on all six instances in that problem set without a solution. With learning, QRSsat solved each of these instances within 5 sec, except for one, which was solved in 24 sec. Thus also in the case of the game problem, learning of conflict clauses was important.

The comparatively large run times of the QBF solvers for the robot navigation problems could be attributed to the increased size of the QBF formulas. A direct approach to Q-ALL SAT seems to be more effective. However, that reason does not apply to the game problems since the increase in size of the QBF formulas is not significant. Learning of  $S$ -conflict clauses is an essential improvement.

## 7 Summary and Ongoing Work

This paper defines the problem Q-ALL SAT, identifies a special class and its complexity, and describes the solution algorithm QRSsat. The method includes a scheme for learning clauses. We have reported first results for the instances of two nontrivial problem classes. The computational results show that the direct attack on Q-ALL SAT with the solver QRSsat is substantially faster for the two problem classes when compared with the performance of three state-of-art QBF solvers on QBF versions of the problem classes.

Q-ALL SAT ignores any costs associated with the values selected for the  $Q$  variables. Applications typically give rise to such costs, and thus lead to variations of Q-ALL SAT. Some applications even give rise to quantified logic problems at the third level of the polynomial hierarchy. Here, too, some versions involve costs. For details about these problems and heuristic solution algorithms, see Truemper [2004]. In ongoing research, we are aiming for exact solution algorithms which handle large application classes effectively.

## Acknowledgements

We thank Charles Otwell who implemented a first version of the solver QRSsat.

## References

- [1997] Bayardo Jr., R. J., Schrag, R. C.: Using CSP look-back techniques to solve real world SAT instances. Proceedings of the 14th National Conference on Artificial Intelligence (1997) 203–208.
- [2000] Bonet B., Geffner, H.: Planning with Incomplete Information as Heuristic Search in Belief Space. Proceedings of the 5th International Conference on AI Planning and Scheduling (2000) 52–61.
- [2003] Castellini, C., Giunchiglia, E., Tacchella, A.: SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. Artificial Intelligence 147 (2003) 85–117.
- [2003] 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.
- [2000] Cimatti, A., Roveri, M.: Conformant Planning via Symbolic Model Checking. Journal of Artificial Intelligence Research 13 (2000) 305–338.
- [1995] Eiter, T., Gottlob, G.: The Complexity of Logic-based Abduction. Journal of the Association for Computing Machinery 42 (1995) 3–42.
- [2002] Giunchiglia, E., Narrizano, M., Tacchella, A.: Learning for Quantified Boolean Logic Satisfiability. Proceedings of the 18th National Conference on Artificial Intelligence (2002).
- [2003] Giunchiglia, E., Narrizano, M., Tacchella, A.: Backjumping for Quantified Boolean Logic satisfiability. Artificial Intelligence 145(1) (2003) 99–120.
- [1999] João, P., Marques-Silva, J. P., Sakallah, K. A.: GRASP – A Search Algorithm for Propositional Satisfiability. IEEE Transactions on Computers 48(5) (1999) 506–521.
- [1995] Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Information and Computation 117(1) (1995) 12–18.
- [2002] Letz, R.: Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. U. Egly and C.G. Fermüller (Eds.): TABLEAUX 2002, LNAI 2381, (2002) 160–175.
- [2003] Plaisted, D., Biere, A., Zhu, Y.: A Satisfiability Procedure for Quantified Boolean Formulae. Discrete Applied Mathematics 130(2) (2003) 291–328.

- [2002] QBFLIB - The Quantified Boolean Formulas Satisfiability Library. <http://www.qbflib.org> (2002).
- [2001] Rintanen, J.: Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formula. International Conference on Logic Programming, Artificial Intelligence and Reasoning (2001) 362–376.
- [1999] Rintanen, J.: Constructing conditional plans by a theorem prover. Journal of Artificial Intelligence 10 (1999) 323–352.
- [2003] SAT 2003, Sixth International Conference on Theory and Applications of Satisfiability Testing, QBF Evaluation, <http://www.satlive.org/QBFEvaluation/index.jsp> (2003).
- [1976] Stockmeyer, L. J.: The polynomial hierarchy. Theoretical Computer Science 3 (1976) 1–22.
- [1999] Straach, J., Truemper, K.: Learning to Ask Relevant Questions. Artificial Intelligence 111 (1999) 301–327.
- [2004] Truemper, K.: Design of Intelligent Systems. Wiley (2004).
- [1997] Zhang, H.: SATO: An Efficient Propositional Prover. Proceedings of the International Conference on Automated Deduction (1997).
- [2001] Zhang, L., Madigan, C. F., Moskewicz M. H., Malik, S.: Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. Proceedings of the International Conference on Computer Aided Design (2001).
- [2002] Zhang, L., Malik, S.: Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. Proceedings of 8th International Conference on Principles and Practice of Constraint Programming (2002).
- [2002] Zhang, L., Malik, S.: Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. Proceedings of the International Conference on Computer Aided Design (2002).