An Algorithm to Evaluate Quantified Boolean Formulae *

Marco Cadoli, Andrea Giovanardi, Marco Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
email: (cadoli|giovanardi|schaerf)@dis.uniroma1.it

Abstract

The high computational complexity of advanced reasoning tasks such as belief revision and planning calls for efficient and reliable algorithms for reasoning problems harder than NP. In this paper we propose Evaluate, an algorithm for evaluating *Quantified Boolean Formulae*, a language that extends propositional logic in a way such that many advanced forms of propositional reasoning, e.g., reasoning about knowledge, can be easily formulated as evaluation of a QBF. Algorithms for evaluation of QBFs are suitable for the experimental analysis on a wide range of complexity classes, a property not easily found in other formalisms. Evaluate is based on a generalization of the Davis-Putnam procedure for SAT, and is guaranteed to work in polynomial space. Before presenting Evaluate, we discuss all the abstract properties of QBFs that we singled out to make the algorithm more efficient. We also briefly mention the main results of the experimental analysis, which is reported elsewhere.

Introduction

Interest in algorithms for the SAT problem has been constant in the AI community. SAT is obviously relevant to AI, and, being the prototypical NP-complete problem, challenges our ability to cope with large knowledge bases. Usage of algorithms for SAT for reasoning tasks different from classical propositional reasoning has been recently emphasized in the literature. For example, real-world problems such as constraint-based planning can be encoded in SAT (Kautz & Selman 1996), and theorem provers for modal logic can use SAT solvers as black boxes (Giunchiglia & Sebastiani 1996).

Anyway, optimizing algorithms for SAT is not enough for the goals of Knowledge Representation: theoretical analysis showed that advanced forms of reasoning such as belief revision, non-monotonic reasoning, reasoning about knowledge, and STRIPS-like planning, have computational complexity higher that the complexity of SAT, cf. e.g., (Eiter & Gottlob 1992) which shows Σ_2^p- and PSPACE-complete reasoning problems. This calls for efficient and reliable algorithms for reasoning problems harder than NP.

In this paper we propose Evaluate, an algorithm for evaluating a *Quantified Boolean Formula* (QBF). Intuitively, QBFs extend propositional logic in a way similar to the extension from first- to second-order logic: in QBFs propositional variables can be quantified over, either existentially, or universally. As an example, $\forall x_1 \exists x_2 \ (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ means "for each truth assignment to x_1 there exists a truth assignment to x_2 such that $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ is true". The above QBF is indeed true: if $x_1 =$ true then x_2 can be assigned to false; if $x_1 =$ false then x_2 can be assigned to true; in both cases $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ is true. The *evaluation problem* for a QBF is to decide whether a given QBF is true or not. QBFs are both harder to cope with, and much more expressive, than pure propositional logic. In fact, many advanced forms of reasoning such as reasoning about knowledge using modal logics, temporal and description logics, can be easily formulated as evaluation of a QBF.

Evaluation of QBFs is similar to SAT also because it is the prototypical problem complete for an important complexity class, i.e., PSPACE. Other reasoning problems, e.g., satisfiability of modal formulae in the system K, are PSPACE-complete too, but syntactically restricted QBFs offer complete problems for other complexity classes relevant to KR such as Σ_2^p and Σ_3^p. Therefore, algorithms for evaluation of QBFs are very suitable for the experimental analysis on a wide range of complexity classes, a property not easily found in other formalisms.

To the best of our knowledge, the only published algorithm for evaluation of QBFs appears in (Büning, Karpinski, & Flögel 1995), and is based on resolution. Evaluate is based on a generalization of the Davis-Putnam (DP) procedure for SAT, and is guaranteed to work in polynomial space. Implementations of DP are still among the most efficient complete algorithms for SAT. Like DP, Evaluate exploits the idea of performing unit propagation as much as possible, and resorts to branching when all other simplifying rules fail. Of course, the different nature of evaluation of QBF and SAT forced us to use specific rules in the design of Eval-

uate, e.g., variables bound by the external quantifier must be dealt with before others. In particular, some rules do not affect soundness and/or completeness, but rather efficiency of the algorithm. As an example, if a QBF has a non-tautologous clause in which all literals are universally quantified, then it is false.

We implemented the algorithm in C++ and performed an extensive analysis of its performances for the evaluation of randomly generated QBFs. The analysis, which is reported in (Cadoli, Giovanardi, & Schaerf 1997), is the first of its kind. In particular, we were able to find patterns for *shift of crossover point* (the point at which half of the instances evaluate to true), *phase transition* (sharp differences between instances close to the crossover point from instances far from it), and *easy-hard-easy* distribution. Some of the result generalize those shown in (Selman, Mitchell, & Levesque 1996) for SAT, while others do not. Moreover, the richer structure of QBFs raises the possibility of analyzing experimental behavior for parameters that do not have any counterpart in the propositional case (e.g., number of quantifiers alternations in a QBF).

The purpose of this paper is to present Evaluate. To this end we also discuss all the properties of QBFs that we singled out to make it more efficient. We also briefly mention the main results of the experimental analysis.

Quantified Boolean Formulae

A QBF has the form

$$Q_1 x_1 \cdots Q_n x_n E(x_1, \ldots, x_n) \qquad (1)$$

where E is a propositional formula involving the propositional variables $x_1, \ldots, x_n$ and every Q_i $(1 \leq i \leq n)$ is either an existential quantifier $\exists$ or a universal one $\forall$. The expression $\exists x_i \phi$ is an abbreviation for "there exists a truth assignment to x_i such that ϕ is true". Analogously, $\forall x_i \phi$ is an abbreviation for "for each truth assignment to x_i, ϕ is true". Inverting quantifiers in a QBF may change its truth value. As an example, inverting quantifiers in $\forall x_1 \exists x_2 \ \ (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ yields $\exists x_1 \forall x_2 \ \ (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$, which is indeed false (cf. Introduction).

Given a generic QBF, we can group in the same set all consecutive variables having the same quantifier. In such a format, each quantifier is applied to a set of variables rather than to a single propositional variable. Moreover, the sequence of quantifiers alternates: an existential quantifier follows a universal quantifier and vice-versa.

A *kQBF* (with k constant integer) is a QBF in which the quantifiers are applied to k disjoint sets of variables and the sequence of quantifiers alternates; sometimes we will add a subscript denoting the type of the most external quantifier in the formula. For example, if X_1, X_2, and X_3 are mutually disjoint sets of propositional variables, then the formula $\exists X_1 \forall X_2 \exists X_3 \ E(X_1, X_2, X_3)$ is a $3\text{QBF}_\exists$. SAT coincides with the evaluation problem for $1\text{QBF}_\exists$. Evaluating a QBF is inherently more difficult than deciding satisfiability of a propositional formula: the evaluation problem for a QBF is PSPACE-complete, and plays the role of prototypical problem for such a class. The problem of evaluating a $k\text{QBF}_\exists$ is Σ_k^p-complete, whereas the problem of evaluating a $k\text{QBF}_\forall$ is Π_k^p-complete, Σ_k^p and Π_k^p being the classes at the k-th level of the Polynomial Hierarchy.

In the rest of this paper we refer to QBFs in *Conjunctive Normal Form* (CNF), i.e., QBFs of the form (1) in which the boolean formula E is a conjunction of *clauses*, each one being a disjunction of *literals* —a negated or non-negated variable. In this case E is called *matrix* of the formula. A QBF is said to be in *hCNF* if every clause contains exactly h literals.

Considering QBF in CNF is not a restriction, since the problem of evaluating such formulae is still PSPACE-complete. With respect to kQBFs, the evaluation problem of kQBFs in 3CNF is complete for the same complexity class of the general case if the most internal quantifier is an existential. As a consequence, in the following we will consider kQBFs of the form

$$Q_1 X_1 \cdots \exists X_k \ \ E(X_1, \ldots, X_k) \qquad (2)$$

in which $E(X_1, \ldots, X_n)$ is in CNF, the most internal quantifier is existential and the sequence of quantifiers alternates. Fixing the type of the most internal quantifier makes the use of subscripts in, e.g., $k\text{QBF}_\exists$ useless. As an example, a 2QBF has the form $\forall X_1 \exists X_2 \ E(X_1, X_2)$, while a 3QBF has the form $\exists X_1 \forall X_2 \exists X_3 \ E(X_1, X_2, X_3)$.

We now show some relevant properties of QBFs that are of interest in the development of the algorithm.

Definition 1 *Given a kQBF of the form (2), we define the sets* Σ *and* Π *as follows:*

$$\Sigma = X_k \cup X_{k-2} \cup \cdots$$

$$\Pi = X_{k-1} \cup X_{k-3} \cup \cdots$$

that is, Σ *is the union of all the sets of existentially quantified variables, while* Π *collects all the universally quantified variables.*

Example 1 *Let F be the QBF*

$$\begin{aligned} \forall W \exists Z \forall X \exists Y \quad & [(x_1 \vee \neg y_1 \vee z_1) \wedge (y_1) \wedge \\ & (w_1 \vee \neg w_2 \vee \neg z_1) \wedge (\neg x_2) \wedge \\ & (y_2 \vee \neg w_1 \vee \neg x_1) \wedge (\neg y_2 \vee z_1)] \end{aligned}$$

Then $\Pi = W \cup X = \{w_1, w_2, x_1, x_2\}$, *while* $\Sigma = Z \cup Y = \{z_1, y_1, y_2\}$.

Using the sets Σ and Π, we can partition the matrix $E(X_1, \ldots, X_k)$ of any kQBF of the form (2) into three matrices:

1. $H(\Sigma)$, containing the clauses in which only variables of Σ occur;
2. $G(\Pi)$, containing the clauses in which only variables of Π occur;
3. $L(\Sigma, \Pi)$, containing the remaining clauses.

Therefore, any such kQBF can be rewritten as:

$$Q_1 X_1 \cdots \exists X_k \ [H(\Sigma) \wedge G(\Pi) \wedge L(\Sigma, \Pi)] \qquad (3)$$

Example 2 *Referring to Example 1:*

$$\begin{aligned} H(\Sigma) &\equiv (y_1) \wedge (\neg y_2 \vee z_1) \\ G(\Pi) &\equiv (\neg x_2) \\ L(\Sigma, \Pi) &\equiv (x_1 \vee \neg y_1 \vee z_1) \wedge (w_1 \vee \neg w_2 \vee \neg z_1) \wedge \\ &\quad (y_2 \vee \neg w_1 \vee \neg x_1) \end{aligned}$$

Lemma 1 (Trivial falsity on Π) *A QBF F of the form (3) is false if $G'(\Pi) \neq \emptyset$, where $G'(\Pi)$ is obtained from $G(\Pi)$ by deleting all tautological clauses.*

Proof. Let us assume that $G'(\Pi)$ is non-empty. Then there exists at least one non tautological clause $C = l_1 \vee \cdots \vee l_m$, where $l_1, \ldots, l_m$ are literals. The truth assignment $l_i = false$ for $i = 1, \ldots, m$ makes G false. Since all variables corresponding to the literals $l_1, \ldots, l_m$ are universally quantified, F is false. □

Notice that the formula in Example 1 is trivially false since $G'(\Pi)$ is non empty $(\neg x_2)$. The simple check required in Lemma 1 can be accomplished in time linear in the size of the matrix. As a consequence, in all the interesting cases we always have that $G(\Pi) = \emptyset$. In the following we assume that this is the case. Thus, the generic form of a QBF becomes:

$$Q_1 X_1 \cdots \exists X_k \ [H(\Sigma) \wedge L(\Sigma, \Pi)] \qquad (4)$$

Lemma 2 (Trivial falsity on Σ) *A QBF F of the form (4) is false if $H(\Sigma)$ is unsatisfiable.*

Proof. If $H(\Sigma)$ is unsatisfiable, there are no truth assignments to the existentially quantified variables that can make the matrix true. Therefore, F is false. □

A corollary of Lemma 2 is that, if a unit clause occurs in $H(\Sigma)$, then it is useless to try to falsify it. In other words, the unit propagation rule of the Davis-Putnam algorithm is also applicable to QBFs when applied to variables in Σ. On the contrary, unit propagation is ruled out for variables in Π by Lemma 1. In fact, if there exists a unit clause whose only variable belongs to Π then the QBF is false.

Example 3 *Let F be the following QBF:*

$$\begin{aligned} F \ = \forall X \exists Y \ & [y_2 \wedge (\neg y_1 \vee y_2 \vee y_3) \wedge (\neg y_2 \vee y_1) \wedge \\ & (x_1 \vee x_2 \vee y_3) \wedge (\neg y_1 \vee x_1 \vee \neg x_2)] \end{aligned}$$

F is false: in fact, $H(Y) \equiv y_2 \wedge (\neg y_1 \vee y_2 \vee y_3) \wedge (\neg y_2 \vee y_1)$ must be satisfiable (Lemma 2), and therefore, $y_2 = true$. By unit propagation $y_1 = true$ and F can be simplified as

$$\forall X \exists Y [(x_1 \vee x_2 \vee y_3) \wedge (x_1 \vee \neg x_2)]$$

Now there exists a clause $(x_1 \vee \neg x_2)$ with all variables in Π. Thus, by Lemma 1, F is false.

Lemma 3 (Trivial truth) *A QBF F of the form (4) is true if $H(\Sigma) \wedge L'(\Sigma)$ is satisfiable, where $L'(\Sigma)$ is obtained from $L(\Sigma, \Pi)$ by deleting all variables in Π.*

Proof. All models of $L'(\Sigma)$ are also models of $L(\Sigma, \Pi)$. Let us assume $H(\Sigma) \wedge L'(\Sigma)$ is satisfiable, and M is one of its models. M makes $H(\Sigma) \wedge L(\Sigma, \Pi)$ true for any truth assignment to the letters in Π. Therefore, F is true. □

Example 4 *The following QBF*

$$\begin{aligned} \forall W \exists Z \forall X \exists Y \ & [(\neg w_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee y_2 \vee z_3) \wedge \\ & (\neg x_2 \vee w_1 \vee \neg z_1) \wedge (x_1 \vee z_2 \vee y_1) \wedge \\ & (\neg z_2 \vee \neg y_2 \vee y_1)] \end{aligned}$$

is trivially true. In fact, $H(\Sigma) \wedge L'(\Sigma) \equiv y_1 \wedge (y_2 \vee z_3) \wedge (\neg z_1) \wedge (z_2 \vee y_1) \wedge (\neg z_2 \vee \neg y_2 \vee y_1)$ is satisfiable since the model $y_1 = true$, $y_2 = true$, $z_1 = false$ satisfies it.

Lemma 3 presents a sufficient condition which is not necessary. For example, the QBF $\forall X \exists Y [(x_1 \vee y_1) \wedge (\neg x_1 \vee \neg y_1)]$ is true while $L'(\Sigma) \equiv y_1 \wedge \neg y_1$ is unsatisfiable. Notice that verifying the condition of the above lemma requires to perform a satisfiability test. Nevertheless, our experimental analysis has shown that the presence of this test makes the algorithm Evaluate more efficient.

There are other conditions that can help us to simplify QBFs. Following the terminology used for the Davis-Putnam procedure, we call literal l *monotone* if its complementary literal does not appear in the matrix E of the QBF. Monotone literals are important because we can immediately assign them a value without any need for branching. The truth value we need to assign them is a function of the set to which they belong. In the following, when we assign a truth value to a literal, we implicitly assign the opposite truth value to the complementary literal.

Lemma 4 (Monotone literals in Σ) *Given a QBF F of the form (2) and a monotone literal $l \in \Sigma$, then F is true if and only if $F' = Q_1 X_1 \cdots \exists X_k E'(X_1, \ldots, X_k)$ is true, where $E'(X_1, \ldots, X_k)$ is obtained from $E(X_1, \ldots, X_k)$ by replacing l with true.*

Due to the lack of space we omit this and the following proofs.

Lemma 5 (Monotone literals in Π) *Given a QBF F of the form (2) and a monotone literal $l \in \Pi$, then F is true if and only if $F' = Q_1 X_1 \cdots \exists X_k E'(X_1, \ldots, X_k)$ is true, where $E'(X_1, \ldots, X_k)$ is obtained from $E(X_1, \ldots, X_k)$ by replacing l with false.*

The conditions of Lemmata 4 and 5 can be checked in polynomial time. There is one more simple situation (whose conditions can be checked in polynomial time) where we can avoid considering all assignments to a variable.

Lemma 6 (Forced assignment for Σ) *Let F be a QBF of the form (2) and C a clause in E such that:*

1. *there exists a literal $l \in X_i \subseteq \Sigma$ in C, and*
2. *all other literals of C belong to the set $X_{i+1} \cup X_{i+3} \cup \cdots \cup X_{k-1} \subseteq \Pi$.*

Algorithm Evaluate
Input: a kQBF CNF formula F of the form (2)
Output: *true* if F is true, *false* otherwise.

```
boolean Evaluate (QBF F)
  { while (F contains a tautological clause C)
        remove C from F;
    if (F is a Π-formula) return Π_Evaluate(F);
    else return Σ_Evaluate(F);
  }
```

Figure 1: The Evaluate algorithm

Then the formula F is true if and only if the formula $F' = Q_1 X_1 \cdots \exists X_k E'(X_1, \ldots, X_k)$ is true, where $E'(X_1, \ldots, X_k)$ is obtained from $E(X_1, \ldots, X_k)$ by substituting all occurrences of l with true.

The Algorithm

The algorithm Evaluate is shown in Figure 1. For solving a QBF, Evaluate performs successive simplifications on the original formula, using techniques such as propagation and backtracking, and performing partial evaluation of subformulae obtained in this way. As a matter of fact, Evaluate makes use of two recursive procedures Σ_Evaluate and Π_Evaluate that interact each other and cooperate in evaluating the input formula. Σ_Evaluate is a procedure that works on a QBF in which the most external quantifier is existential (Σ-formulae). Dually, Π_Evaluate works on Π-formulae, that is, formulae in which the most external quantifier is universal. The matrix of both Σ- and Π-formulae is in CNF.

The main procedure Evaluate takes as input a QBF F and returns its truth value. As a matter of fact, it performs two simple actions: first of all, all tautological clauses in F (if any) are removed. Notice that the elimination of tautological clauses can be performed once and for all: in fact, none of the successive manipulations that the algorithm makes on the input formula can create a tautological clause. Then, Evaluate invokes either Σ_Evaluate or Π_Evaluate according to whether F is a Σ-formula or a Π-formula.

Figure 2 shows the procedure Π_Evaluate. The procedure takes in input a Π-formula and returns its truth value; it is at the same time an iterative and recursive procedure, that works as follows.

Base of recursion. First of all, Π_Evaluate checks whether the input formula F is formed by all existentially quantified variables; in this case, F is indeed a 1QBF and then for evaluating it is sufficient to invoke any procedure for SAT (cf. line (1)) –the Davis-Putnam algorithm in our implementation. Successively, Π_Evaluate verifies whether F is *trivially* true by using Lemma 3. To this end, it computes the 1QBF G, obtained by removing from F all the universally quantified variables (cf. (2)), and checks its satisfiability. To check whether G is satisfiable, Π_Evaluate makes use of a procedure for SAT (cf. (3)). If G is satisfiable, the input

Procedure Π_Evaluate
Input: a Π-formula of the form:
$F = \forall X_1 \exists X_2 \cdots \forall X_{k-1} \exists X_k \;\; E(X_1, \ldots, X_k)$
Output: *true* if F is true, *false* otherwise.

```
boolean Π_Evaluate (QBF F)
(1)  { if (all the variables in F are existentially quantified)
         return SAT(F);
(2)    remove from F all the universally quantified
       variables, thus obtaining the formula G;
(3)    if (SAT(G)) return true;
(4)    while (there are uninstantiated variables in X_1)
(5)      { if (E is an empty set of clauses)
(6)            return true;
(7)        if (E contains an empty clause)
(8)            return false;
(9)        if (there is a clause made of all
             universally quantified variables in E)
(10)           return false;
(11)       if (there is a unit clause c formed by
             a literal l ∈ Σ in E)
(12)         { assign true to l;
(13)           simplify F with l = true;
             }
(14)       else if (there is a monotone literal l ∈ Σ in E)
(15)         { assign true to l;
(16)           simplify F with l = true;
             }
(17)       else if (there is a monotone literal l ∈ Π in E)
(18)         { assign false to l;
(19)           simplify F with l = false;
             }
(20)       else if (there is a clause c including a literal
                 l ∈ X_h ⊆ Σ in E and all other literals
                 of c are in X_{h+1} ∪ ··· ∪ X_{k-1} ⊆ Π)
(21)           { assign true to l;
(22)             simplify F with l = true;
               }
(23)       else
(24)           { choose a literal l ∈ X_1;
(25)             QBF OldF = F;
(26)             assign false to l;
(27)             simplify F with l = false;
(28)             if (Π_Evaluate(F) == false)
                   return false;
(29)             else
(30)                 { F = OldF;
(31)                   assign true to l;
(32)                   simplify F with l = true;
(33)                   return Π_Evaluate(F);
                    }
               }
      } /* while */
(34)  return Σ_Evaluate(F);
  }
```

Figure 2: The Π_Evaluate procedure

formula F is true, and then Π_Evaluate can stop and return *true* (cf. (3)). We could use an incomplete, faster algorithm for testing satisfiability of G, but experiments showed that a sound and complete procedure for SAT results in a more efficient evaluation of the QBF.

Iteration and forced assigments. If F is not trivially true, the algorithm proceeds iteratively: during the generic iteration, the formula F is simplified by means of truth values assignments to one or more of its variables. The assignments made by Π_Evaluate can be of two kinds: *forced* and *unforced*. A *forced assignment* to a variable x in F causes F to be simplified in an other QBF G such that F is true *if and only if* G is true. Thus, forced assignments are very important, since they simplify the evaluation process without making backtracking necessary.

Unforced assignments are performed on variables that belong to X_1, that is, the most external set of variables in the input formula F (cf. the input of Π_Evaluate). Unforced assignment generally require the use of backtracking, as we will see later.

Turning back to the algorithm, simplifications on F are performed until one of the following cases is verified:

1. Simplifications made on F cause all the variables in X_1 to be instantiated (cf. (4)); in this case X_1 is empty, and F is indeed a Σ-Formula. For evaluating F, Π_Evaluate invokes the Σ_Evaluate procedure (cf. (34)).
2. Simplifications made on F cause the matrix E to be empty (cf. (5)); this means that all clauses in E have been satisfied, and so F is true. As a consequence, the algorithm stops returning *true* (cf. (6)).
3. Simplifications made on F cause the matrix E to contain an empty clause (cf. (7)); in this case F is trivially false and the algorithm stops returning $false$ (cf. (8)).
4. Simplifications made on F cause the matrix E to contain a clause formed by all universally quantified variables (cf. (9)). By Lemma 1 the formula F is false. So, the algorithm stops returning $false$ (cf. (10)).

If none of the above cases is verified, Π_Evaluate checks whether it is possible to perform forced assignments on variables in F. Using Lemmata 3, 4, 5, 6, and unit propagation (using Lemma 2), we have characterized four different situations where forced assignments can be performed (cf. statements from (11) to (22)).

Branch. If no forced assignments can be performed, Π_Evaluate simplifies the input formula by means of an unforced assignment made on a variable x in X_1 (cf. statements between (24) and (33)). Since there is no way for excluding neither the value *true* nor the value $false$ for x, the evaluation of F is splitted in the evaluation of the two subformulae obtained assigning respectively $false$ and *true* to x. Since x is a universally quantified variable, F is true if and only if both these subformulae are true. The instantiation of x is indeed a branch in the evaluation process of F; for this reason, we call x the *branch variable*, whereas the literals corresponding to x are called *branch literals*.

Three heuristics for choosing the branch literal were adopted. In increasing efficiency, they are: 1) random, 2) try to maximize the number of Horn/dual-Horn clauses after simplification (using on ideas of (Crawford & Auton 1993)), and 3) privilege those variables having the maximum number of occurrences in short clauses and, among them, variables that appear most frequently in the matrix of the input formula (using ideas of the SAT-solver "Böhm" as described in (Buro & Büning 1993)).

After having chosen the branch variable and a particular branch literal (cf. (24)), the actual structure of F is saved for future backtracking (cf. (25)). Successively, Π_Evaluate is recursively invoked on the formula obtained assigning $false$ to the branch literal l (cf. (26), (27) and (28)). If the result of the recursive call of Π_Evaluate is $false$, the procedure can stop and return $false$ (cf. (28)). Otherwise, Π_Evaluate backtracks and restores the old structure of F (cf. (30)). The formula F is then simplified assigning *true* to the branch literal (cf. (31) and (32)). Lastly, the procedure returns the result obtained applying recursively Π_Evaluate on F (cf. (33)).

The procedure Σ_Evaluate is very similar to Π_Evaluate, and so is omitted. As a matter of fact, Σ_Evaluate differs from Π_Evaluate only in case of branch: now X_1 is a set of existentially quantified variables. While in Π_Evaluate a logical AND between the two recursive calls is performed, in Σ_Evaluate is necessary to perform a logical OR between them.

Experimental results

In (Cadoli, Giovanardi, & Schaerf 1997) we present in detail an experimental analysis of the complexity of valuating randomly generated kQBF instances. Here we briefly recall these results together with some new ones. In our tests, we have generated kQBF instances according to two different models: *Fixed Clause Length* (FCL) and *Constant Probability* (CP); both of them are well-known in the literature (cf. e.g., (Selman, Mitchell, & Levesque 1996)).

In the FCL model each formula is generated so that all clauses are different, non-tautologous, and contain exactly h literals, not all of them universally quantified. If V is the set of all propositional variables in the formula (that is $V = X_1 \cup \cdots \cup X_k$), then a clause is produced by randomly choosing h different variables in V and negating each one with probability 0.5. The CP model has the same parameters of the FCL model, except for h, which represents in this case the average number of literals per clause in the formula. The empty clause and unit clauses are disallowed.

The results we obtained can be summarized as follows:

- If clauses are long enough, e.g., 6CNF, evaluation is more difficult for 3QBF than 2QBF, and for 2QBF

than 1QBF.

- Phase transition phenomena for the percentage of true instances as a function of the #clauses/#variables per set ratio exists in all the examinated cases (2QBF-3/4/5/6CNF, 3QBF-3/6CNF) of QBF.
- An easy-hard-easy pattern for the average difficulty of the instances as a function of the #clauses/#variables per set ratio has been observed in the following cases: 2QBF-5/6CNF, and 3QBF-6CNF. In particular, we noticed a correlation between the location of the hardest instances and the crossover point for 2QBF-6CNF and 3QBF-6CNF. The easy-hard-easy pattern, instead, has not been observed in all the other cases (2QBF-3/4CNF, and 3QBF-3CNF)
- The number of clauses at the crossover point is not a linear function of the number of variables (as it was for 1QBF); in all the cases in which we could examine a wide range for the number of variables per set (2QBF-3/4CNF, and 3QBF-3CNF), we have verified that the number of clauses at the crossover point is proportional to the square root of the number of variables per set
- As in the 1QBF case, instances generated according to the CP model are, on average, much easier than those generated by means of the FCL model.
- In all the considered cases, the true instances are the easiest for kQBF with k odd, whereas the easiest instances are the false ones for kQBF with k even.
- Unbalancing the number of existentially-quantified and universally-quantified variables has remarkable effects on the difficulty of kQBF instances; if the $\exists$-variables are increased and the $\forall$-variables are decreased, the instances become much harder with respect to the instances where their number is balanced. An opposite phenomenon has been observed when the $\forall$-variables are increased and the $\exists$-variables are decreased.

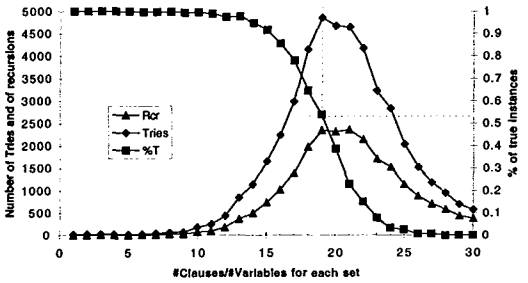

Figure 3: Results for 3QBF-6CNF

To give an example of the results we obtained, we report in Figure 3 the percentage of true instances, the average number of truth values assignment made to the variables before being able to evaluate the formula (tries), and the average number of recursions, as a function of the #clauses/#variables per set ratio. The total number of variables is 30 (10 variables per set), the number of clauses varies from 10 to 300, with a step of 10, and 500 experiments for each setting of the parameters were run. The curves regarding the number of tries and recursions show an easy-hard-easy pattern, with the hardest instances being associated with the crossover point. At the peak, evaluation takes about 10 seconds on average on a 75MHz/32MB SPARC10.

Conclusions and Future Work

In this paper we have presented Evaluate, an algorithm to evaluate quantified boolean formulae. This algorithm can help us in understanding the computational structure of all problems that belong to the classes captured by QBFs, i.e., each level of the Polynomial Hierarchy and PSPACE. We do not claim Evaluate is the best conceivable algorithm, but it can benefit from advancements in technology of algorithms for SAT, because it uses the black-box principle. We singled out some cases which appear in practice to require more computational resources, hence made a first step in the development of algorithms which are efficient in the average case. This is also one step for our long-term goal to characterize, from the experimental point of view, the most widely accepted KR formalisms. In order to substantiate this claim, we are currently experimenting on the usage of our algorithm to solve decision problems that are not in NP, such as satisfiability of modal formulae. To this end, we have found reductions that map a modal formula into a QBF.

References

Büning, H. K.; Karpinski, M.; and Flögel, A. 1995. Resolution for quantified boolean formulas. *Information and Computation* 117:12–18.

Buro, M., and Büning, H. K. 1993. Report on a SAT competition. *EATCS Bulletin* 49:143–151.

Cadoli, M.; Giovanardi, A.; and Schaerf, M. 1997. Experimental analysis of the computational cost of evaluating quantified boolean formulae. In *Proc. of AI*IA-97*, number 1321 in LNAI, 207–218. Springer-Verlag.

Crawford, J. M., and Auton, L. D. 1993. Experimental results on the crossover point in satisfiability problems. In *Proc. of AAAI-93*, 21–27.

Eiter, T., and Gottlob, G. 1992. On the complexity of propositional knowledge base revision, updates and conterfactuals. *Artif. Intell.* 57:227–270.

Giunchiglia, F., and Sebastiani, R. 1996. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of KR-96*, 304–314.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. of AAAI-96*, 1194–1201.

Selman, B.; Mitchell, D.; and Levesque, H. 1996. Generating Hard Satisfiability Problems. *Artif. Intell.* 81:17–29.