A Symbolic Search Based Approach for Quantified Boolean Formulas

Gilles Audemard and Lakhdar Saïs[⋆]

CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{audemard,sais}@cril.univ-artois.fr

Abstract. Solving Quantified Boolean Formulas (QBF) has become an important and attractive research area, since several problem classes might be formulated efficiently as QBF instances (e.g. planning, non monotonic reasoning, two-player games, model checking, etc). Many QBF solvers has been proposed, most of them perform decision tree search using the DPLL-like techniques. To set free the variable ordering heuristics that are traditionally constrained by the static order of the QBF quantifiers, a new symbolic search based approach (QBDD(SAT)) is proposed. It makes an original use of binary decision diagram to represent the set of models (or prime implicants) of the boolean formula found using search-based satisfiability solver. Our approach is enhanced with two interesting extensions. First, powerful reduction operators are introduced in order to dynamically reduce the BDD size and to answer the validity of the QBF. Second, useful cuts are achieved on the search tree thanks to the nogoods generated from the BDD representation. Using DPLL-likes (resp. local search) techniques, our approach gives rise to a complete QBDD(DPLL) (resp. incomplete QBDD(LS)) solver. Our preliminary experimental results show that on some classes of instances from the QBF evaluation, QBDD(DPLL) and QBDD(LS) are competitive with state-of-the-art QBF solvers.

Keywords: Quantified boolean formula, Binary decision diagram, Satisfiability.

1 Introduction

Solving quantifi ed boolean formulas has become an attractive and important research area over the last years. Such increasing interest might be related to different reasons including the fact that many important artifi cial intelligence problems (planning, non monotonic reasoning, formal verifi cation, etc.) can be reduced to QBFs which is considered as the canonical problem of the PSPACE complexity class. Another important reason comes from the recent impressive progress in the practical resolution of the satisfi ability problem.

Many solvers for QBFs have been proposed recently (e.g. [11, 18, 12, 10]), most of them are obtained by extending satisfi ability results. This is not surprising since QBFs

[⋆] This work has been supported in part by the IUT de Lens, the CNRS and the Region Nord/Pas-de-Calais under the TAC Programme is a natural extension of SAT where the boolean variables are universally or existentially quantifi ed. Most of these solvers take a formula in the prenex clausal form as input and are variant of Davis Logemann and Loveland procedures (DPLL) [7]. However, one of the main drawback of such proposed approaches is that variables are instantiated according to their occurrences in the quantifi er prefi x (i.e. from the outer to the inner quantifi er group). Such preset static ordering limits the effi ciency of the search-based QBF solvers. Indeed, in some cases, invalidity of a given QBF might be related to its subparts with variables from the most inner quantifi er groups. Consequently, following the order of the prefi x might lead to a late and repetitive discovery of the invalidity of the QBF.

The main goal of this paper is to set free the solver from the preset ordering of the QBF and to facilitate the extension of the satisfi ability solvers. To this end, binary decision diagrams are used to represent in a compact form the set of models of the boolean formula found by a given satisfi ability solver. It give rise to a new QBF solver QBDD(SAT) combining satisfi ability search based techniques with binary decision diagram. For completeness and effi ciency reasons, our approach is enhanced with two key features. On the one hand, reduction operators are proposed to dynamically reduce at least to some extent the size of the binary decision diagram and to answer the validity of the QBF. On the other hand, for each model found by the satisfi ability search procedure, its prime implicant is represented in the BDD and a nogood is returned from the reduced BDD representation and added to the formula. The approach we present in this paper signifi cantly extends our preliminary results [2]. We give a more general framework with additional features such as prime implicants encoding, cuts generation. Two satisfi ability search techniques (DPLL and local search techniques) are extended to QBF using our proposed approach.
The paper is organized as follows. After some preliminaries and technical background on quantifi ed boolean formulas and binary decision diagram, it is shown how binary decision diagram can be naturally combined with satisfi ability search based techniques to handle QBFs. Using systematic search (respectively stochastic local search) techniques, preliminary experiments on instances of the last QBFs evaluation are presented and show that QBDD(SAT) is competitive and can somtimes achieve signifi cant speedups over state-of-the-art QBF solvers.

2 Preliminaries and technical background

Before introducing our approach, we briefly review some necessary defi nitions and notations about quantifi ed boolean formulas and binary decision diagram.

2.1 Quantified boolean formulas

Let $\mathcal{P}$ be a fi nite set of propositional variables. Then, $\mathcal{L}_{\mathcal{P}}$ is the language of quantifi ed boolean formulas built over $\mathcal{P}$ using ordinary boolean formulas (including propositional constants $\top$ and $\bot$) plus the additional quantifi cation ($\exists$ and $\forall$) over propositional variables.

We consider quantified boolean formula in the prenex form: $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ (in short $QX\Psi$, QX is called the prefix of Φ and Ψ the matrix of Φ) where $Q_i \in \{\exists, \forall\}$, $X_k, \ldots, X_1$ are disjoint sets of variables and Ψ a boolean formula. Consecutive variables with the same quantifier are grouped. The rank of a variable $x \in X_i$ is equal to i (noted $rank(x)$). Variables in the same quantifier group have the same rank value. We define a prefix ordering of QBF formula $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ as the partial ordering obtained according to the decreasing rank of the variables, noted $X_k < X_{k-1} < \cdots < X_1$. A QBF formula Φ is said to be in clausal form if Φ is in prenex form and Ψ is in Conjunctive Normal Form (CNF). Note that we can consider QBFs with inner quantifier Q_1 as existential. Indeed, if Q_1 is a universal quantifier then suppressing $Q_1 X_1$ from the prefix and all occurrences of $x \in X_1$ from the matrix lead to an equivalent QBF. We define $Var(\Phi) = \bigcup_{i \in \{1,\ldots,k\}} X_i$ the set of variables of Φ. A literal is the occurrence of propositional variable in either positive (l) or negative form ($\neg l$). $Lit(\Phi) = \bigcup_{i \in \{1,\ldots,k\}} Lit(X_i)$ the set of complete literals of Φ, where $Lit(X_i) = \{x_i, \neg x_i | x_i \in X_i\}$. We note $var(l)$ the variable associated to a literal l. A literal $l \in \Phi$ is a unit literal iff l is existentially quantified and $\exists c = \{l, l_1, \ldots, l_i\} \in \Psi$ s.t. $\forall l_j, 1 \leq j \leq i, var(l_j)$ is universally quantified and $rank(l_j) < rank(l)$. A monotone literal is defined in the usual way as in the pure boolean case (i.e. l is monotone in Φ iff it appears either positively or negatively).

To define the semantic of quantified boolean formulas, let us introduce some necessary notations. Let S be the set of assignments over the set of variables V. The Up-projection (resp. Down-projection) of a set of assignments S on a set of variables $X \subset V$, denoted $S \uparrow X$ (resp. $S \downarrow X$), is obtained by restricting each assignment to literals in X (resp. in $V \backslash X$). The set of all possible assignments over X is denoted by 2^X. An assignment over X is denoted by a vector of literals $\overrightarrow{x}$. In the same way, Up-projection and Down-projection also apply on vector of literals $\overrightarrow{x}$. If $\overrightarrow{y}$ is an assignment over Y s.t. $Y \cap X = \emptyset$, then $\overrightarrow{y}.S$ denotes the set of interpretations obtained by concatenating $\overrightarrow{y}$ with each interpretation of S. Finally, $\Psi(\overrightarrow{x})$ denotes the boolean formula Ψ simplified with the partial assignment $\overrightarrow{x}$. An assignment $\overrightarrow{x}$ is an implicant or a model (resp. nogood) of Ψ; noted $\overrightarrow{x} \models \Psi$ (resp. $\overrightarrow{x} \not\models \Psi$) iff $\Psi(\overrightarrow{x}) = \top$ (resp. $\Psi(\overrightarrow{x}) = \bot$). An implicant (resp. nogood) $\overrightarrow{x}$ is called prime implicant (resp. minimal nogood) of Ψ iff $\nexists \overrightarrow{y} \subset \overrightarrow{x}$ s.t. $\overrightarrow{y} \models \Psi$ (resp. $\overrightarrow{y} \not\models \Psi$).

A QBF formula is valid (is true) if there exists a solution (called a total policy) defined as follows. It is a simplified version of the definition by Sylvie Coste-Marquis *et al.* [13].

Definition 1. *Let $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ a quantified boolean formula and $\pi = \{\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n\}$ a set of models of the boolean formula Ψ. π is a total policy of the quantified boolean formula Φ iff π recursively verifies the following conditions:*

1. *$k = 0$, and $\Psi = \top$*
2. *if $Q_k = \forall$, then $\pi \uparrow X_k = 2^{X_k}$, and $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ is a total policy of $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$*
3. *if $Q_k = \exists$, then $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ and $\pi \downarrow \overrightarrow{x}_k$ is a total policy of $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$*

Remark 1. Let π be a total policy of $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$. If $Q_k = \forall$ then we can rewrite π as $\bigcup_{\overrightarrow{x}_k \in 2^{X_k}} \{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$ and if $Q_k = \exists$, then $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ and π can be rewritten as $\{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$

Example 1. Let $\Phi = \exists x_5 x_6 \forall x_2 x_4 \exists x_1 x_3 \Psi$ be a QBF formula, where $\Psi = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_4 \vee x_3) \wedge (x_5 \vee x_6)$. Φ is a valid QBF, since the set of models $\pi = \{(\neg x_5, x_6, x_2, x_4, \neg x_1, x_3)$,$(\neg x_5, x_6, x_2, \neg x_4, \neg x_1, x_3)$, $(\neg x_5, x_6, \neg x_2, \neg x_4, x_1, \neg x_3)$, $(\neg x_5, x_6, \neg x_2, x_4, x_1, x_3)\}$ is a total policy of Φ (Figure 1). The different projection operations are illustrated as follow :
$\pi \uparrow \{x_5, x_6\} = \{(\neg x_5, x_6)\}$
$\pi' = \pi \downarrow \{x_5, x_6\}$
$= \{(x_2, x_4, \neg x_1, x_3), (x_2, \neg x_4, \neg x_1, x_3), (\neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_2, x_4, x_1, x_3)\}$
$\pi' \uparrow \{x_2, x_4\} = \{(\neg x_2, \neg x_4), (\neg x_2, x_4), (x_2, \neg x_4), (x_2, x_4)\} = 2^{\{x_2, x_4\}}$

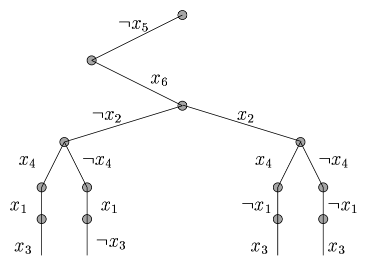


Fig. 1. Policy decision tree representation (example 1)

Motivated by the impressive results obtained in practical solving of the satisfi ability problem, several QBF solvers have been developed recently. Most of them are extensions of the well known DPLL procedure including many effective SAT results such as learning, heuristics and constraint propagation (QUBE [11], QUAFFLE [18], EVALUATE [6], DECIDE[16]). For examples, QUBE [11] and QUAFFLE [18] extend backjumping and learning techniques, EVAUATE[6] and DECIDE [16] extend some SAT pruning techniques such as unit propagation.

Algorithm 1 gives a general scheme of a basic DPLL procedure for checking the validity of QBFs. It takes as input variables of the QBF prefi x $< X_k, \ldots, X_1 >$ associated to quantifi ers $Q_k, \ldots, Q_1$ and a matrix Ψ in clausal form. It returns true if the QBF formula $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ is valid and false otherwise. The algorithm starts by simplifying the formula using unit propagation and monotone literal rules. Then, if the current simplifi ed matrix contains the empty clause then the current QBF is invalid (value false is returned); otherwise, if the current matrix is empty then the QBF formula is valid (value true is returned). The next step consists in choosing the next variable to instantiate (splitting rule) in the most external non empty set of variables X_k. This differ from the DPLL satisfi ability version, since variables are instantiated according to their prefi x ordering. Depending on the quantifi er Q_k of the chosen variable, left and/or right branchs are generated. If $Q_k = \forall$ (resp. $Q_k = \exists$) the right branch is generated only if the value returned in left branch is true (resp. false). The search tree developped by the QBF DPLL procedure can be seen as an and/or tree search.

Algorithm 1: *QDPLL* for QBF

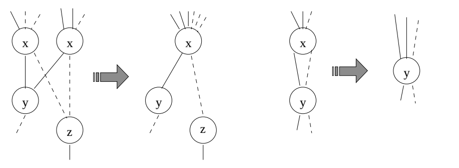
```
Data   : $\Psi$ : matrix of the QBF; $< X_k, \dots, X_1 >$ : prefix of the QBF $\Phi$
Result : true if the QBF $\Phi$ s valid, false otherwise
begin
    Simplify($\Psi$);
    if $\emptyset \in \Psi$ then return false;
    if $\Psi = \emptyset$ then return true;
    if $X_k = \emptyset$ then return QDPLL($\Psi, < X_{k-1}, \dots, X_1 >$);
    choose (by heuristic) a literal $l \in X_k$;
    if (($Q_k = \forall$) and QDPLL($\Psi \cup \{l\}, < X_k - \{l\}, \dots, X_1 >$)=false) then
        return false;
    if (($Q_k = \exists$) and QDPLL($\Psi \cup \{l\}, < X_k - \{l\}, \dots, X_1 >$)=true) then
        return true;
    return QDPLL($\Psi \cup \{\neg l\}, < X_k - \{l\}, \dots, X_1 >$);
end
```

One of the major drawback of the extension of the DPLL procedure to QBF concerns the imposed prefi x ordering. Such restrictive ordering might lead to performance degradation of the QBF solver. Since, good ordering might be lost. Furthermore, such limitation makes diffi cult the extension of certain interesting results obtained on the satisfi ability problem (see for example [8] for random problem or [14] for structured problems). One can cite, stochastic local search another search paradigm widely used in SAT (e.g. [17]) that received little attention in QBF. A fi rst integration of local search in QBF solver (WalkQSAT) has been investigated in [10]. WalkQSAT is an implementation of conflict and solution directed backjumping in QBF. It uses a local search solver to guide its search.

2.2 Binary decision diagram

A Binary Decision Diagram (BDD) [1, 5] is a rooted directed acyclic graph with two terminal nodes that are referred to as the 0-terminal and the 1-terminal. Every non-terminal node is associated with a primary input variable such that it has two outgoing edges called the 0-edge corresponding to assigning the variable a false truth value, and the 1-edge corresponding to assigning the variable a true truth value. An Ordered Binary Decision Diagram (OBDD) is a BDD such that the input variables appear in a fi xed order on all the paths of the graph, and no variable appears more than once in the path. A Reduced Ordered BDD (ROBDD) is an OBDD that results from the repeated application of the following two rules:

1. Share all equivalent sub-graphs (Figure 2.a).
2. Eliminate all redundant nodes whose outgoing edges point to the same node (Figure 2.b).



a. Sharing sub-formulas b. Redundant node

Fig. 2. Reduction Rules

A (RO)BDD representing a boolean formula Ψ is noted $(RO)BDD(\Psi)$.

Figure 3 illustrates the ROBDD representation of the policy π ($ROBDD(\pi)$) shown in example 1 where the solid edges denote the 1-edges and the dashed edges denote the 0-edges. The ROBDD order is the same as the prefi x ordering of variables ($\{x_5, x_6\} < \{x_2, x_4\} < \{x_1, x_3\}$) of the QBF Φ.

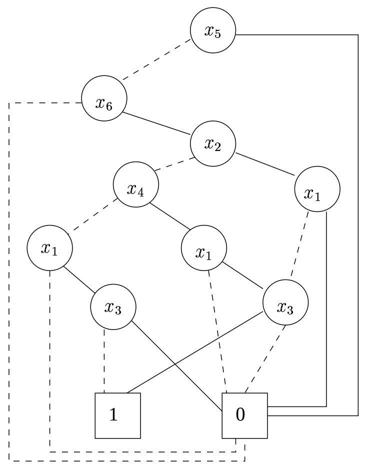


Fig. 3. BDD representation of a policy (example 1)

ROBDDs have some interesting properties. They provide compact and unique representation of boolean functions, and there are effi cient algorithms performing all kinds of logical operations on ROBBDs. For example, it is possible to check in constant time whether an ROBDD is true or false. Let us recall that for boolean formula, such problem is NP-complete. Despite the exponential size of the ROBDD in the worst case, ROBDD is one of the most used data structure in practice.

In the rest of this paper, only reduced ordered BDDs are considered and for short we denote them as BDDs. For aquantifi ed boolean formula, the order used in the ROBDD follows the prefi x ordering of the QBF.

3 QBDD(SAT): a symbolic search based approach

In this section, to make the QBF solver freed from the preset ordering of the variables (i.e. fixed by the QBF prefix), we propose an original combination of classical SAT solver with binary decision diagrams. In figure 4, we give a general scheme of our symbolic search based approach QBDD(SAT). More precisely, to check the validity of a QBF $\Phi = QX\Psi$, our approach makes use of a satisfiability technique to search for models (*SAT enumerator*) of the boolean formula Ψ. For each found model m, a prime implicant pi is extracted (*Compute PI*) and disjunctively added to the BDD (*bdd = or(bdd, pi)*) using the prefix ordering of the variables. If the current set of prime implicants represent a total policy then its BDD representation is reduced to 1-terminal node (see section 3.1) and the QBDD(SAT) answer the validity of Φ. As was mentioned earlier, QBDD(SAT) can be instantiated with any satisfiability search technique. For example, QBDD(DPLL) (resp. QBDD(LS)) refer to a QBF solver obtained by instantiating SAT enumerator with DPLL-like (resp. local search) techniques. At the end of the satisfiability search process, if the BDD is not reduced to a 1-terminal node (i.e. a total policy is not found) then depending on the completeness of the SAT used enumerator QBDD(SAT) return either invalid or unknown. Consequently, the QBDD(SAT) is complete iff the satisfiability used solver is also complete.

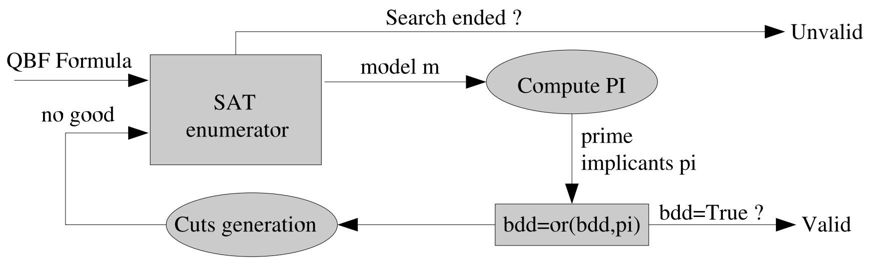


Fig. 4. QBDD(SAT): general scheme

For space complexity reason, only prime implicants of the boolean formula are encoded in the BDD, nogoods found during the satisfiability search process are not considered. Paths to 0-terminal node in the generated BDD do not represent the nogoods of the boolean formula. Consequently, the 0-terminal node and its incoming edges can be omitted. However, to reduce the search space, for each model (or prime implicant) encoded in the BDD a new nogood is generated and added to the boolean formula (see section 3.2).

3.1 Quantifier reductions operators

To reduce the BDD size and to answer the validity of the QBF, additional reduction operator is given in Figure 5. If a node x is existentially quantified and one of its child nodes is the 1-terminal node then any reference to the node x is simply replaced by a reference to its 1-terminal node. We call such reduction operation *existential reduction*. Interestingly enough, when x is universally quantified and its two child nodes are 1-terminal, such node is eliminated using the classical BDD node reduction (Figure 2.b).

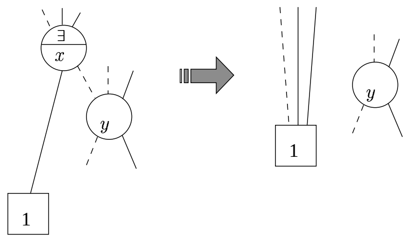


Fig. 5. Existential Reduction

During the BDD construction process, in addition to classical reduction operations 2, existential reduction is applied. If the set of models represents a total policy of the QBF formula then the BDD built from such models is reduced to a 1-terminal node as stated by the following property :

Property 1. Let $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ be a QBF formula and $\pi = \{\overrightarrow{x}_1, \overrightarrow{x}_2, \ldots, \overrightarrow{x}_n\}$ a set of models of Ψ. If π is a total policy of Φ then the BDD(π) is reduced to the 1-terminal node.

Proof. The proof is obtained by induction on k. For $k = 0$, the BDD representing the constant $\top$ is a 1-terminal node (by definition of a total policy). Suppose that the property holds for $k - 1$, let us prove that it holds for k. By definition of a total policy two case are considered :

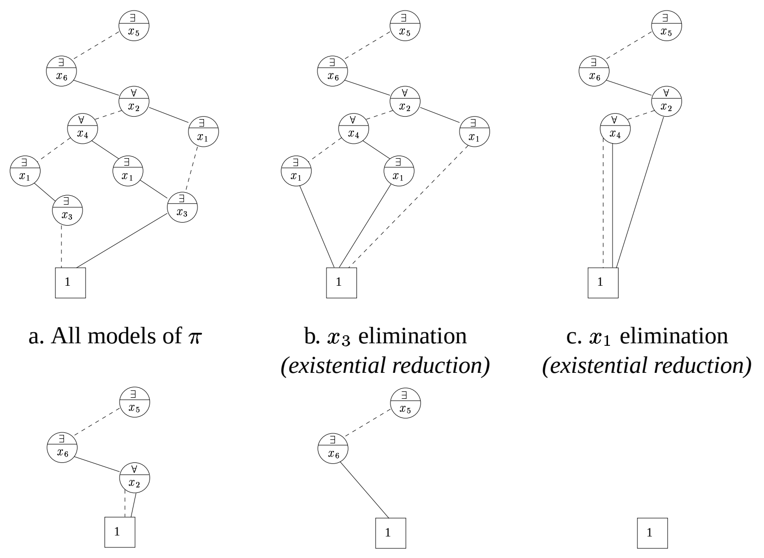
1. if $Q_k = \forall$, then $\pi \uparrow X_k = 2^{X_k}$, and $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ is a total policy of the QBF formula $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$. By induction hypothesis, we can deduce that BDD($\pi \downarrow \overrightarrow{x}_k$) is reduced to 1-terminal node. Consequently all the leaf of the BDD(2^{X_k}) are 1-terminal nodes. Then by repeatedly applying the Redundant node rule on such a BDD, we obtain a BDD reduced to a 1-terminal node.
2. if $Q_k = \exists$, then $\pi \uparrow \overrightarrow{X}_k = \{\overrightarrow{x}_k\}$ and $\pi \downarrow \overrightarrow{x}_k$ is a total policy of the QBF formula $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$. By induction hypothesis BDD($\pi \downarrow \overrightarrow{x}_k$) is a 1-terminal node. Consequently, the BDD($\{\overrightarrow{x}_k\}$) can be seen as a branch ended on a 1-terminal node. Repeatedly applying the existential reduction rule, the BDD(π) is reduced to a 1-terminal node.

Remark 2. Dually, *universal reduction* can also be defined. As 0-terminal node of the BDD represents undefined state, then universal reduction operator can not be used in our approach.

The following example shows the dynamic reduction of the BDD associated with the set of models representing the total policy of the QBF given in the example 1.

Example 2. Let Φ be the QBF formula of the example 1 and π its associated policy. The figure 6 represents the reduction phase of the BDD(π) representation:

- The figure 6.a is a BDD representation of the policy π (only paths with final 1-terminal node are represented).
- The existential reduction rule allows the x_3 elimination (figure 6.b) and the x_1 elimination (figure 6.c).
- The redundant node reduction rule allows the x_4 elimination (figure 6.d) and the x_2 elimination (figure 6.e).
- Finally the existential reduction rule suppresses x_5 and x_6 and the bdd of the policy is restricted to the 1-terminal node representing the true formula (figure 6.f) and proving that π is a total policy.



a. All models of π | b. x_3 elimination *(existential reduction)* | c. x_1 elimination *(existential reduction)*

d. x_4 elimination *(redundant node reduction)* | e. x_2 elimination *(redundant node reduction)* | f. x_5, x_6 elimination *(existential reduction)*

Fig. 6. BDD reduction of a QBF total policy

3.2 Generating cuts from BDD

In order to avoid search of models belonging to different total policies, we introduce in the following different possible cuts (nogoods) that can be generated from the model, prime implicant or from the BDD under construction.

Definition 2. *Let $\Phi = Q_k X_k, \ldots, Q_2 X_2, Q_1 X_1 \Psi$ a QBF s.t. $Q_2 = \forall$, $Q_1 = \exists$ and $\overrightarrow{x}$ is a model of Ψ. We define, $nogood_m(\overrightarrow{x}) = \bigvee\{\neg l | l \in \overrightarrow{x} \downarrow X_1\}$ as the nogood* *obtained from* $\overrightarrow{x}$. *In the same way we define* $nogood_{pi}(\overrightarrow{pi})$ *as the nogood extracted from a prime implicant* $\overrightarrow{pi}$.

Obviously, if $\overrightarrow{pi}$ is a prime implicant obtained from a model $\overrightarrow{x}$ then $nogood_{pi}(\overrightarrow{pi}) \models nogood_m(\overrightarrow{x})$.
Using the example 1, we show in figure 7, that for a model $\overrightarrow{x} = \{\neg x_5, x_6, \neg x_2, x_4, x_1, x_3\}$ the $nogood_m(\overrightarrow{x}) = (x_5 \vee \neg x_6 \vee x_2 \vee \neg x_4)$ ovoid useless search for models of different policies. Considering $\overrightarrow{pi} = \{x_6, x_1, x_3\}$ a prime implicant of $\overrightarrow{x}$, we can generate a strong cut $nogood_{pi}(\overrightarrow{pi}) = \neg x_6$. Interestingly enough, thanks to reductions defined above, the BDD encoding such a prime implicant is reduced to a 1-terminal node and the validity of the QBF is answered.

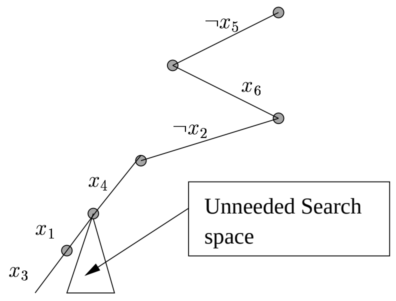


Fig. 7. cuts generation

Let us recall, that when a prime implicant is disjunctively added to the BDD under construction, the reduction operators eliminate the variables of the inner existential quantifier group X_1. In addition, universally quantified variables in X_2 can be eliminated, when there two outgoing edges point to 1-terminal node. Such a reduction process is iterated recursively. Consequently, to generate strong cuts, in practice each time a prime implicant is added to the BDD, a nogood is extracted from the new reduced BDD and added to the boolean formula.

3.3 QBDD(DPLL) approach

The algorithm 2 represents the QBF solver obtained by a combination of DPLL procedure with BDD. As we can see, the algorithm search for all models until the BDD representation (characterized by the global variable bdd initialized to the 0-terminal node) is reduced to a 1-terminal node, in that case the algorithm terminates and answers the validity of the QBF formula. If the algorithm backtrack to level 0, then the QBF formula is invalid. In other case search continues (back is returned). The function $Simplify()$ enforces the well known unit propagation process. While the function $conflictAnalysis()$ implements learning scheme used by the most efficient satisfiability solvers. Function $primeImplicants()$ extracts a prime implicant from a model I of the boolean formula Ψ. Computing a prime implicant from a given model can be done in linear time. Indeed, for each literal l of the given model m, we verify if the $m \backslash \{l\}$ is also a model of Psi, in such a case the literal l is deleted from m. Finally, $cutsGeneration()$ generates from the current bdd a new nogood and add it to the cnf Ψ (see section 3.2). Example 3 gives a possible trace of QBDD(DPLL) algorithm.

Algorithm 2: Combining BDD and DPLL : QBDD(DPLL)

```
Data   : $\Psi$ : set of clauses; X=$\{X_k, \ldots, X_1\}$ the prefix set of the QBF;
         $I$ a partial interpretation ; $d$ level in the search tree, initially set to 0
Result : $valid$ if the QBF is valid, $invalid$ otherwise;
         $back$ is returned to continue the search for other models.
begin
    Simplify($\Psi$);
    if $\emptyset \in \Psi$ then
        conflictAnalysis();
        return $back$;
    if $\Psi = \emptyset$ then
        $pi := primeImplicants(I, \Psi)$;
        $bdd$ := or($bdd$, $pi$);
        if equal(bdd,1-terminal) then return $valid$;
        cutsGeneration($pi$,bdd);
        return $back$;
    Let $l \in X_i$ $(i \in \{1 \ldots k\})$ be the chosen branching variable;
    if (QBDD(DPLL) ($\Psi \cup \{l\}, X - \{l\}, I \cup \{l\}, d+1$)=$valid$) or QBDD(DPLL)
    ($\Psi \cup \{\neg l\}, X - \{l\}, I \cup \{\neg l\}, d+1$)=$valid$) then
        return $valid$;
    if ($d = 0$) then
        return $invalid$;
    return $back$;
end
```

Example 3. Let us consider the formula Φ of example 1. Suppose that the algorithm QBDD(SAT) starts the search by assigning x_1 and x_5. At this step, the clause $(\neg x_4 \vee x_3)$ is satisfi ed. If we assign a truth value to the literal $\neg x_4$ then a fi rst model $\{x_5, \neg x_4, x_1\}$ is found and is added to the *bdd*. This variable is reduced to a single path to the 1-terminal node : $< x_5, \neg x_4 >$ (variable x_1 is deleted by existential reduction). The clause $(\neg x_5 \vee x_4)$ is added to the formula Ψ. A backtrack is done to search for other models and the assignment of the literal x_4 implies x_3 using unit propagation. A second model $\{x_5, x_4, x_1, x_3\}$ is added to the bdd which becomes reduced to the 1-terminal node using existential and redundant reduction operators. So, search ends and returns the validity of the formula Φ.

3.4 QBDD(LS) approach

One of the important features of our QBDD(SAT) is that any satisfi ability search based technique can be used without any major adaptation. Particularly, local search techniques can be integrated in a simple way. Using the state-of-the-art local search WalkSat, we obtain a new incomplete solver QBDD(LS). It answers that the QBF formula is valid, when the set of found models represents a total policy (see property 1); otherwise the solver returns unkown. Our QBDD(LS) solver differs from WalkQSat [10] in the sence that our approach uses local search as a model generator instead of using it to guide the DPLL search procedure.

4 Empirical Evaluation

The experimental results reported in this section are obtained on a Pentium IV 3 GHz with 1GB RAM, and performed on a large panel (644 instances) of the QBF'03 evaluation instances [3]. Theses instances are divided into different families (`log`, `impl`, `toilet`, `k_*`...). For each instance cpu time (in seconds) limit is set to 600 seconds. The QBDD(DPLL) solver is based on chaff like solver called minisat [9], and the QBDD(LS) solver is based on walksat.

4.1 Behaviour of QBDD(DPLL) solver

Figure 8 presents a pictorial view on the behaviour of different versions of the QBDD(DPLL) solver :

- QBDD(DPLL) is the basic version without computing prime implicants and without generating cuts from the BDD
- QBDD(DPLL) +CUTS is augmented with the generation of cuts from BDD
- QBDD(DPLL) +PI computes prime implicants from a given sat model
- QBDD(DPLL) +PI+CUTS contains these two key features.

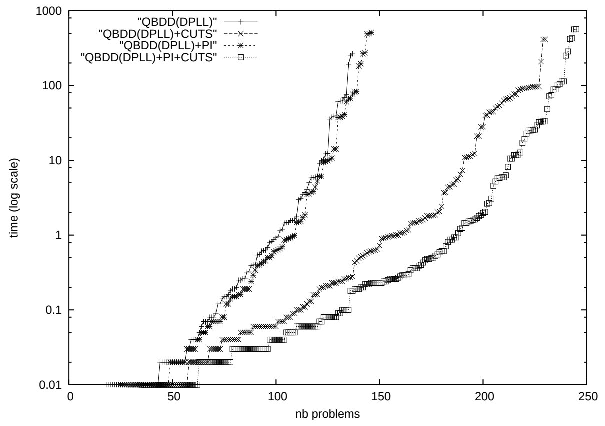


Fig. 8. Number of instances solved vs CPU time

The plot in figure 8 is obtained as follows: the x-axis represents the number of benchmarks solved and the y-axis (in log scale) the time needed to solve this number of problems. This figure exhibits clearly that the basic version is the less effective one.

Adding only prime implicants does not signifi cantly improve the basic version. However, generating cuts from BDD produces a real improvements, the number of solved problems in less than 600 seconds increases from 130 to 220. Finally, the best version of the QBDD(DPLL) solver is otained by ading cuts and by encoding prime implicants instead of models. Table 1 gives some explanations about these results, it shows the number of models needed to answer the validity of different instances. Since only necessary models are computed with the generation of cuts, the number of models is smaller than without computing cuts. The generation of prime implicants produces some improvements because each prime implicants includes a large potential part of the qbf policies.

problem	valid	QBDD(DPLL)	QBDD(DPLL) +PI	QBDD(DPLL) +CUTS	QBDD(DPLL) +PI+CUTS
impl06	Y	6112	6012	2142	550
k_poly_n-1	Y	>45000	>45000	1850	862
toilet_a_10_01.5	N	41412	41412	22486	20 917

Table 1. number of models

Table 2 exhibits the number of instances solved with respect to the size of the quantifi er prefi x. The method seems to be very effi cient with 2QBF formulas and seems to be be quite inefi ccient with large prefi x (for example only 11.6% of problems with prefi x greater than 8 are solved). However, QBDD(DPLL) is not restricted to 2QBF instances since it solves half of 5-QBF instances. This result agree with those obtained in [15] on 2-QBF using an original combination of two satisfi ability solver.

prefix size	2	3	4	5	6	7	≥ 8
nb problems	57	339	8	34	7	22	154
nb solved	51	148	1	19	1	11	18
percent	89.5	43.6	12.5	55.8	14.2	50	11.6

Table 2. solved instances wrt prefix size

4.2 Comparison with state of the art solvers

We compare our solvers QBDD(SAT) to state of the art QBF solvers QUBE [11] and QUANTOR [4]. Here, QBDD(DPLL) solver exploits cuts and prime implicants. Table 3 gives the cpu time comparison between these three solvers on different QBF families instances. *#N* represents the number of problems in the family, *#S* the number of problem solved by a given solver and *TT* the total cpu time needed for a solver to solve all instances of the family. If a method fails to solve an instance then 600 seconds are added to the total cpu time.

Worst results of QBDD(DPLL) are obtained on `toilet` families since only 106 instances are solved in less than 600 seconds, whereas QUBE and QUANTOR solve all quite easily. It's the same for the `k_*` family. Best results are obtained on the `robot` family. Since QBDD(DPLL) solves more and fastly instances than the two other solvers. Furthermore, QBDD(DPLL) solves hard instances of the QBF'03 evaluation for the fi rst time. On a lot of families (`z4`, `flipflop`, `log`) results of all solvers are comparable.

		Qube		Quantor		QBFBDD	
family	#N	#S	TT	#S	TT	#S	TT
robots	48	36	7 214	35	7 970	42	4 270
k_*	171	98	44 813	99	43 786	32	83 658
flipfbp	7	7	0.36	7	1.2	7	3.2
toilet	260	260	40	260	256	106	93 860
impl	8	8	0.01	8	0.02	6	1211
tree-exa10	6	6	0.07	6	0.01	6	1.7
chain	8	8	497	8	0.3	2	4183
tree-exa2	6	6	0.01	6	0.01	1	3000.1
carry	2	2	0.23	2	0.69	2	0.71
z4	13	13	0.06	13	0.08	13	0.1
blocks	3	3	0.2	3	0.47	3	3.2
log	2	2	18.4	2	42	2	31

Table 3. CPU time comparison beetwen Qube, Quantor and QBDD(DPLL)

4.3 Comparison with QBDD(LS)

Since QBDD(LS) is an incomplete solver which can not prove the invalidiy of qbf instances, it is quite difficult to make a fair comparison with other qbf solvers. We present its preliminary behaviour on some valid instances and make short comparison with QBDD(DPLL), `Quantor` and `Qube`. Table 4 gives the time to solve an instance and the number of models computed during search (if available). For QBDD(LS), each instance is solved 20 times and the median is reported.

	QBDD(LS)		QBDD(DPLL)		Quantor		Qube	
problem	model	time	model	time	model	time	model	time
impl10	3440	8	2673	2	-	0.01	-	0.01
toilet_c_04_10.2	2850	14	?	>600	-	0.01	-	0.01
robots_1_5_2_3.3	58	6	79	0.28	-	453	-	0.7
comp.blif_0.10_1.00_0_1_out_exact	3205	308	4647	1.8	-	0.03	-	>600
tree_exa10-20	?	>600	1095	0.2	-	0.01	-	0.1

Table 4. comparison on valid instances

These preliminary experiments shows that QBDD(LS) is quite promising. It solves 76 of the 150 valid instances. It is competitive on some QBF instances and obtain better results than other solvers on some other instances.

5 Conclusion

In this paper, a new symbolic search based approach for QBF is presented. It makes an original combination of model search satisfiability techniques with a binary decision diagrams. BDD are used to encode prime implicants of the boolean formula. Reduction operators that prevent to some extent the blowup of the BDD size and answer the validity of the QBF are presented. Interestingly enough, nogoods are generated from the reduced BDD allowing strong cuts in the SAT search space. The main advantage of our approach is that it is freed from any ordering of the variables. This facilitates the extension of satisfiability solver to deal with quantified boolean formulas. Two satisfiability search paradigms (systematic and local search) have been investigated giving rise to two QBF solvers (QBDD(DPLL) and QBDD(LS)). Experimental results on instances from the QBFs evaluation show the effectiveness of our approach. More interestingly, some open hard QBF instances are solved for the first time.

References

1. S. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27:509–516, 1978.
2. Gilles Audemard and Lakhdar Sais. Sat based bdd solver for quantified boolean formulas. In *proceedings of the 16th IEEE international conference on Tools with Artificial Intelligence*, pages 82–89, 2004.
3. Daniel Le Berre, Laurent Simon, and Armando Tachella. Challenges in the qbf arena: the sat'03 evaluation of qbf solvers. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, *LNAI*, pages 452–467, 2003.
4. Armin Biere. Resolve and expand. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2004.
5. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–692, C-35.
6. Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, Madison (Wisconsin - USA), 1998.
7. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.
8. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI–01)*, August 4–10 2001.
9. Niklas Eén and Niklas Sörensson. An extensible sat solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, p. 502–508, 2003.
10. Ian P. Gent, Holger H. Hoos, Andrew G. D. Rowley, and Kevin Smyth. Using stochastic local search to solve quantified boolean formulae. In *Proceedings of the 9th international conference of principles and practice of constraint programming*, pages 348–362, 2003.
11. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, Italy, June 2001.
12. Reinhold Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux 2002*, pages 160–175, Copenhagen, Denmark, 2002.
13. Sylvie Coste Marquis, Helene Fargier, Jerome Lang, Daniel Le Berre, and Pierre Marquis. Function problems for quantified boolean formulas. Technical report, CRIL - France, 2003.
14. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC01)*, 2001.
15. Darsh Ranjan, Daijue Tang and Sharad Malik Niklas. A Comparative Study of 2QBF Algorithms. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 2004.
16. Jussi Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae. In *Proceedings of the First International Conference on Quantified Boolean Formulae (QBF'01)*, pages 84–93, 2001.
17. Bart Selman, Hector Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.
18. Lintao Zhang and Sharad Malik Towards a symetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 200–215, 2002.