Solving QBF by Clause Selection*

Mikoláš Janota[1] and **Joao Marques-Silva**[1,2]
[1] INESC-ID, IST, Lisbon, Portugal
[2] University College Dublin, Ireland
mikolas@sat.inesc-id.pt, jpms@tecnico.ulisboa.pt

Abstract

Algorithms based on the enumeration of implicit hitting sets find a growing number of applications, which include maximum satisfiability and model based diagnosis, among others. This paper exploits enumeration of implicit hitting sets in the context of Quantified Boolean Formulas (QBF). The paper starts by developing a simple algorithm for QBF with two levels of quantification, which is shown to relate with existing work on enumeration of implicit hitting sets, but also with recent work on QBF based on abstraction refinement. The paper then extends these ideas and develops a novel QBF algorithm, which generalizes the concept of enumeration of implicit hitting sets. Experimental results, obtained on representative problem instances, show that the novel algorithm is competitive with, and often outperforms, the state of the art in QBF solving.

1 Introduction

The enormous success of SAT enabled us to solve problems that go beyond the NP complexity class. Among these, Quantified Boolean Formulas (QBFs) represent an important formalism. Indeed, as deciding QBFs is PSPACE-complete, they cover a wide range of problems. Namely from model checking, planning, or two-player games [Papadimitriou, 1994; Benedetti and Mangassarian, 2008; Rintanen, 2007; Narodytska *et al.*, 2014].

In the recent years, a number of approaches were developed to solve QBF. Among the most representative are *conflict/solution driven learning* [Cadoli *et al.*, 1998; Rintanen, 1999; Zhang and Malik, 2002; Giunchiglia *et al.*, 2001], which extends SAT clause learning, and a number of approaches that *expand* the given QBF into a SAT problem [Biere, 2004; Benedetti, 2004; Janota *et al.*, 2012]

This paper proposes a novel approach for QBF solving that has two distinctive features: 1) it uses a SAT solver as an NP oracle, 2) it is anchored in the duality hitting set principle. In the Boolean CNF world, the duality hitting set principle lets us reason about *minimally unsatisfiable sets (MUSes)* by traversing *maximally satisfiable sets (MSSes)*. This has numerous applications in MUS computation (cf. [Reiter, 1987; Liffiton and Sakallah, 2008]) and more recently in MaxSAT solving and other domains [Karp, 2010; Chandrasekaran *et al.*, 2011; Davies and Bacchus, 2011; Stern *et al.*, 2012; Moreno-Centeno and Karp, 2013; Davies and Bacchus, 2013a; Previti and Marques-Silva, 2013; Davies and Bacchus, 2013b; Slaney, 2014].

We first relate MUS computation to solving QBF with two levels of quantification ($\forall\exists$). Then, we extend this idea for arbitrary number of quantification levels. Hence, the proposed approach is not only a novel approach to solving QBF but it can also be seen as an extension of the duality hitting set principle for PSPACE.

The proposed approach instantiates a SAT solver at each quantification level, whose task is to select or deselect clauses at that level. The fact that we are using a SAT solver as the underlying reasoning engine in our algorithm is a big engineering advantage. Indeed, if a better SAT solver appears, it can simply substitute the old one in our implementation. The main contributions of the paper are summarized as follows:

(1) A novel algorithm for solving QBF, called QESTO, is developed. (2) A link is established to implicit hitting set algorithms. (3) A link is established to the CEGAR-based QBF solving. (4) A prototype was implemented and evaluated.

The paper begins by overviewing the concepts and notation used (Sec. 2) and continues by developing an algorithm for two-level QBF (Sec. 3). Sec. 4 generalizes this approach to general QBF and Sec. 4.2 relates to other concepts and algorithms. Finally Sec. 5 describes experimental evaluation and Sec. 6 concludes the paper.

2 Preliminaries

Throughout the paper we assume an infinite set of Boolean variables, commonly denoted as x, u, e, etc. Boolean connectives ($\wedge$,$\vee$,$\neg$) are used under the traditional semantics. An assignment to a set of Boolean variables X is a mapping from X to the Boolean constants $\{0,1\}$. For an assignment $\tau : X \to \{0,1\}$ and a formula ϕ we write $\phi|_\tau$ for the *application* of τ to the formula ϕ. Hence $\phi|_\tau$ is obtained by replacing all occurrences of each variable $x \in X$ by the

*This work is partially supported by SFI PI grant BEACON (09/IN.1/I2618), FCT grant POLARIS (PTDC/EIA-CCO/123051/2010), CMU-Portugal grant AMOS CMUP-EPB/TIC/0049/2013, and national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

constant $\tau(x)$ and any trivial simplifications are performed ($y \vee 1 \equiv 1$, etc.). An assignment τ is a *satisfying assignment* for a formula ϕ if $\phi|_\tau = 1$. A formula is called *satisfiable* if there exists a satisfying assignment for it and otherwise it is *unsatisfiable*. A satisfying assignment of a formula ϕ is often referred to as a *model* of ϕ. For two formulas ϕ and ψ we write $\phi \models \psi$ if any satisfying assignments of ϕ is also a satisfying assignment of ψ.

A Boolean formula is called a *literal* if it is a Boolean variable or its negation. For a literal l, we write $\mathtt{var}(l)$ for the variable in l. A Boolean formula is called a *clause* if it is a disjunction of literals. A Boolean formula is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses. As common in literature, we often treat clauses as sets of literals (implicitly disjoined) and formulas in CNF as sets of clauses (implicitly conjoined). Observe that the empty set of clauses is semantically true (as the empty conjunction is true) and the empty clause is semantically equivalent to false (as the empty disjunction is false).

Quantified Boolean formulas (QBFs) [Giunchiglia *et al.*, 2009] extend Boolean formulas by allowing to quantify over variables using the existential or the universal quantifier ($\exists$, $\forall$). Any QBF can be rewritten to an equivalent Boolean formula by rewriting any $\forall x.\, \Phi$ as $\Phi|_{\{x\mapsto 0\}} \wedge \Phi|_{\{x\mapsto 1\}}$ and $\exists x.\, \Phi$ as $\Phi|_{\{x\mapsto 0\}} \vee \Phi|_{\{x\mapsto 1\}}$.

A QBF is *closed* if all variables are bound by a quantifier. A QBF is in *prenex form* if it can be written as $Q_1 x_1 \dots Q_n x_k.\, \phi$ where ϕ is a Boolean formula and $Q_i \in \{\exists, \forall\}$. For formulas in prenex form, the quantifier part is called the *prefix* and the Boolean part the *matrix*. Unless specified otherwise, in this paper we will be dealing with formulas that are both prenex and closed and the matrix is in CNF. Any adjacent variables with the same quantifier in the prefix are merged into *blocks*. This enables us to write QBFs in the form $Q_1 X_1 \dots Q_n X_k.\, \phi$ where X_i are sets of variables and $Q_i \neq Q_{i+1}$ is assumed. We say that X_i is a block at the *quantification level* i. We write $\mathsf{lv}(x)$ for the *quantification level* of a variable x, i.e., $\mathsf{lv}(x) = i$ if $x \in X_i$. We extend this notation to literals. If a variable $\mathsf{lv}(x) = i$ and $Q_i = \forall$ than we say that x is universal and analogously for existential variables. The class of closed QBF formulas in prenex form with CNF matrix is referred to as PQCNF.

We model a SAT solver by the function $\mathtt{SAT}(\psi)$, which returns a satisfying assignment if the given formula ψ is satisfiable; otherwise it returns the constant $\bot$.

An important fact about PQCNF formulas is that any variables universally quantified at the end of the prefix can be removed. For instance, $\exists e_1 e_2 \forall u.\, (e_1 \vee u) \wedge (\neg e_2 \vee \neg u)$ is equivalent to $\exists e_1 e_2.\, e_1 \wedge \neg e_2$.

3 Solving Two-level PQCNF

Consider a formula $\forall X \exists Y.\, \phi$, where ϕ is in CNF. To motivate our discussion, let us investigate what must happen for the formula to be *false*. There must be an assignment τ to the variables X so that there is *no* corresponding value for the Y variables that would make the matrix ϕ true. In other words, the CNF $\phi|_\tau$ is *unsatisfiable*. In the opposite case, when the formula is *true*, any assignment τ to X makes the CNF $\phi|_\tau$ satisfiable. Hence, to decide whether a given 2-level PQCNF formula is true or not, it is sufficient to decide whether there exists such assignment to X or not. This is the approach we will take from now on.

Algorithm 1: Two-level Selection QCNF Solving

```
   input : ∀X∃Y. φ
1  output:truth value
   𝒞 ← {(¬s_C ∨ ¬l) | C ∈ φ, l ∈ C, l universal}
2  while true do
3      τ ← SAT(𝒞)            // compute selection
4      if τ = ⊥ then return true
5      S ← {C ∈ φ | τ(s_c) = 1}
6      φ' ← { ⋁_{l∈C, var(l)∈Y} l | C ∈ S }
7      μ ← SAT(φ')                // find a model
8      if μ = ⊥ then return false
9      S' ← {C ∈ φ | there is l ∈ C s.t. μ(l) = 1}
10     𝒞 ← 𝒞 ∧ ⋁_{C∈φ, C∉S'} s_C
```

Now let us look more closely at the effect of the application of τ to ϕ. Consider the simple example $\forall u \exists e.\, \{(u \vee e), (u \vee \neg e), (\neg u \vee e)\}$. If $\tau(u) = 1$, the application of τ yields the CNF $\{(e)\}$ and $\tau(u) = 0$ yields the CNF $\{(e), (\neg e)\}$. We can see that the application of τ acts as a *selector* on the clauses comprising of existential variables. In this particular case, $\tau(u) = 0$ selects an unsatisfiable set of clauses. This resembles the way clauses are selected in recent approaches for implicitly manipulating hitting sets [Chandrasekaran *et al.*, 2011; Davies and Bacchus, 2011; Stern *et al.*, 2012; Moreno-Centeno and Karp, 2013; Previti and Marques-Silva, 2013]. However, in QBF, clauses cannot be selected arbitrarily. Consider another simple example $\forall u \exists e.\{(u \vee e), (\neg u \vee \neg e)\}$. Here, regardless of the value of u, the clauses are never selected at the same time, i.e., the result is always satisfiable. This is because selecting a clause requires setting to false all of its universal literals.

The above ideas give rise to a natural algorithm. First, select some set of clauses S and test for its satisfiability. If it is unsatisfiable, stop. If it is satisfiable, block S and any of its subsets to be selected in the future and then repeat. The blocking step can be improved by looking at the satisfying assignment μ of S. Such assignment satisfies some set S' s.t. $S \subseteq S'$. Hence, we can block any set that is a subset of S'.

This idea is made precise by Algorithm 1. We consider a formula $\mathscr{C}$, which captures the requirements on the considered sets of clauses. Sets of clauses are modeled by *selection variables* s_C, which have the traditional meaning that a clause $C \in \phi$ is selected whenever s_C is true. The formula $\mathscr{C}$ is initialized by the condition that a clause can be selected only if it is possible to set all of its universal literals to false.

The algorithm iterates over the following sequence of steps. First it chooses some set of clauses that is permitted by $\mathscr{C}$ (ln. 3). If no such set exists, the algorithm terminates and returns true (the formula is true because there is no set of clauses yielding unsatisfiability). Next, the algorithm tests whether the existential part of the selected clauses are satisfiable (ln. 7). If it is unsatisfiable, then we are done because τ is

an assignment to the universal variables that selects an unsatisfiable set of clauses. If not, we block all subsets of clauses satisfied by the model. This is done by calculating the set S' of clauses satisfied by the assignment to the existential variables and then requiring that at least one clause from its complement is selected.

Example 1. *Consider the following matrix with the prefix $\forall u_1 u_2 u_3 \exists e_1 e_2 e_3$; the clauses are identified by a number on the left and they are graphically split into the universal and existential part:*

$$\begin{array}{lccc} ① & u_1 \vee u_2 & \vee & e_1 \vee e_2 \\ ② & u_1 \vee u_3 & \vee & \neg e_1 \vee e_2 \\ ③ & \neg u_1 \vee u_2 & \vee & e_3 \\ ④ & \neg u_2 \vee u_3 & \vee & \neg e_1 \vee e_3 \end{array}$$

Let us begin by selecting the clauses ① and ②, i.e., $s_1 = s_2 = 1$. This is possible because it is possible to set to false all the literals u_1, u_2, u_3. The selected clauses correspond to the existential part $\phi' = \{e_1 \vee e_2, \neg e_1 \vee e_2\}$, which is satisfiable by the assignment $\mu = \{e_1 \mapsto 0, e_2 \mapsto 1, e_3 \mapsto 0\}$, let's say. The assignment μ satisfies the clauses ①, ②, and ④. Hence, the clause ③ must be selected in any further attempts, i.e., the condition $\mathscr{C}$ is strengthened by the unit clause (s_3). Due to this condition, in the next iteration the algorithm cannot *select the clause ① nor ② since they contain the literal u_1 but ③ contains the literal $\neg u_1$. Let's say the algorithm selects ③ and ④, which yields the existential part $\phi' = \{e_3, \neg e_1 \vee e_3\}$, satisfiable by $\mu = \{e_1 \mapsto 0, e_2 \mapsto 0, e_3 \mapsto 1\}$, which satisfies the clauses ②, ③, and ④. This means that the clause ② must be selected in any next attempts, which is a contradiction with the previous condition because ② and ③ cannot be selected at the same time due to the respective literals u_1 and $\neg u_1$. Hence the algorithm terminates and responds that the formula is true.*

4 Generalizing for Arbitrary Prefixes: QESTO

Now we look at how the ideas for solving PQCNF by selection generalize to prefixes of arbitrary length. Hence, we assume that we are given a formula of the form $Q_1 X_1 \dots Q_n X_n . \phi$, where $Q_i \in \{\exists, \forall\}$, $Q_n = \exists$, and ϕ is a propositional formula in CNF. Recall that $Q_n = \exists$ can be assumed as any universal variable at the end of the prefix can be removed.

For the sake of presentation, we adopt the *game-theoretic* point of view on QBF (cf. [Goultiaeva *et al.*, 2011]). Any QBF in prenex closed form is seen as a game between the *universal* and *existential player*. The universal player tries to falsify the matrix while the existential tries to satisfy it. Under this perspective, a QBF is true if and only if there is a *winning strategy* for the existential player.

More precisely, a game consists of n rounds and in round i, the player Q_i assigns a value to all the variables X_i. At the end of the game we evaluate the matrix ϕ by the accumulated assignment. The game is won by the universal player if the matrix evaluates to false, and it is won by the existential player if it evaluates to true. We say that there is a winning strategy for the existential player if the player can play so that he wins any game irregardless of how the universal player plays. Analogously, we define a winning strategy for the universal player. A formula is true if and only if there is a winning strategy for the existential player. Conversely, formula is false if and only if there is a winning strategy for the universal player.

Example 2. *Consider the formula $\forall u \exists e . \{\neg u \vee e, u \vee \neg e\}$. The assignments $\{u \mapsto 0\}$,$\{e \mapsto 1\}$ represent a game where the existential player loses. The assignments $\{u \mapsto 1\}$,$\{e \mapsto 1\}$ represent a game where the existential player wins. There exists a winning strategy for the existential player, which is $\{e \mapsto u\}$. Hence, the formula is true.*

Example 3. *Consider the formula $\exists e_1 \forall u \exists e_2 . \{e_1 \vee u \vee e_2, e_1 \vee u \vee \neg e_2, \neg e_1 \vee \neg u \vee e_2, \neg e_1 \vee \neg u \vee \neg e_2\}$. The assignments $\{e_1 \mapsto 0\}$, $\{u \mapsto 0\}$,$\{e_2 \mapsto 1\}$ represent a game where the existential player loses. There exists a winning strategy for the universal player, which is $\{u \mapsto e_1\}$. Hence, the formula is false.*

The game-theoretic perspective lets us look at the selection-based solving symmetrically: the existential player tries to deselect clauses—so that the matrix becomes true—and the universal tries to select clauses—so that the matrix becomes false.

The following concepts concretize this idea. For each clause $C \in \phi$ and a quantification level $k \in 1..n$ we introduce a variable s^k_C, which indicates that the clause C is *selected at the quantification level k*. If s^k_C is false, then we say that C is *deselected at the quantification level k*. Selecting and deselecting clauses is guided by the following rules:

1. A clause can be selected at level k if it was selected in all the previous levels and all its literals at the level k are set to false.
2. A clause can be deselected at level k if it was deselected in one of the previous levels *or* at least one of the literals at level k is set to true.

Observe that deselection behaves monotonically, i.e., once a clause is deselected it cannot be selected back again. This is unsurprising as once a clause is satisfied during the game, it remains satisfied.

The next thing to consider is how to encode the rules of the game. These are formulated as follows:

1. If all clauses $C \in \phi$ are deselected at some level k, the universal player loses.
2. If a clause $C \in \phi$ does not contain any literal l with $\mathsf{lv}(l) > k$ then the existential player loses if the clause is selected at level k.

These constraints correspond to the rules discussed above but they detect as soon as possible that a player has lost, i.e., once all clauses are satisfied, the universal player loses, and once the empty clause is derived, the existential player loses.

In its workings, the proposed algorithm is similar to the two-level one. Starting from the first level, the players select and deselect clauses based on the rules of the game. At each level i we maintain a condition $\mathscr{C}_i$, which blocks choices that lead to a loss of player Q_i. These conditions are strengthened throughout the course of the algorithm. If a condition $\mathscr{C}_i$

Algorithm 2: QESTO: Solving PQCNF

1 **for** $i \in 1..n$ **do** $\mathscr{C}_i \leftarrow$ `InitCondition`(i)
2 $\mathsf{qlev} \leftarrow 1$
3 $S_0 \leftarrow \phi$
4 **while** *true* **do**
5 **if** $\mathsf{qlev} = n+1$ **then**
6 $\mathsf{conflict} \leftarrow \{s_C^n \mapsto 0 \mid C \in \phi\}$
7 **Goto** 11
8 $\alpha \leftarrow \bigwedge_{C \in S_{\mathsf{qlev}-1}} s_C^{\mathsf{qlev}-1} \wedge \bigwedge_{C \notin S_{\mathsf{qlev}-1}} \neg s_C^{\mathsf{qlev}-1}$
9 $(\mu, \mathsf{conflict}) \leftarrow \mathtt{SAT}(\mathscr{C}_{\mathsf{qlev}} \wedge \alpha)$
10 **if** $\mu = \bot$ **then**
11 $(\mathsf{btlev}, C_L) \leftarrow \mathtt{Analyze}(\mathsf{qlev}, \mathsf{conflict})$
12 **if** $\mathsf{btlev} = -1$ **then return** $(Q_{\mathsf{qlev}} = \forall)$
13 $\mathscr{C}_{\mathsf{btlev}} \leftarrow \mathscr{C}_{\mathsf{btlev}} \wedge C_L$
14 $\mathsf{qlev} \leftarrow \mathsf{btlev}$
15 **else**
16 $S_{\mathsf{qlev}} = \left\{ C \in \phi \mid \mu(s_C^{\mathsf{qlev}}) = 1 \right\}$
17 $\mathsf{qlev} \leftarrow \mathsf{qlev} + 1$

Algorithm 3: Condition Initialization

Function `InitCondition` (k)
if $k = 1$ **then** $\mathscr{C}_1 \leftarrow \left\{ s_C^1 \triangleq \bigwedge_{l \in C, \mathsf{lv}(l)=1} \neg l \mid C \in \phi \right\}$
else $\mathscr{C}_k \leftarrow \left\{ s_C^k \triangleq s_C^{k-1} \wedge \bigwedge_{l \in C, \mathsf{lv}(l)=k} \neg l \mid C \in \phi \right\}$
if $Q_k = \forall$ **then** $\mathscr{C}_k \leftarrow \mathscr{C}_k \wedge \bigvee_{C \in \phi} s_C$
else
  for $C \in \phi$ **do**
  if *there is no existential* $l \in C$ *s.t.* $\mathsf{lv}(l) > k$ **then**
  $\mathscr{C}_k \leftarrow \mathscr{C}_k \wedge s_C^k$
return $\mathscr{C}_k$

becomes unsatisfiable, it means that player Q_i has *lost* under the current choices. Hence, we perform an analysis that tries to rectify some of the previous choices of the losing player. This consists in strengthening one of the conditions $\mathscr{C}_k$ for some $k < i$ (we look more closely at the loss analysis later).

We name the algorithm QESTO (*Qbf clausE SelecTion sOlver*), which is listed in Algorithm 2. The algorithm uses the subroutine `InitCondition` (Algorithm 3) to initialize the conditions $\mathscr{C}_i$, $i \in 1..n$. Then QESTO continues by iterating the main loop (lns. 4–15) until the formula is solved. In each iteration the algorithm operates at a quantification level qlev. It is constructing sets S_i, which consist of the clauses currently selected at level i. To simplify the presentation, we also introduce the set S_0, which contains all the clauses in ϕ. To construct the set S_{qlev} we search for a model of the condition $\mathscr{C}_{\mathsf{qlev}}$ restricted by the selections made so far (ln. 9). If this SAT call results in unsatisfiability, we backtrack to a quantification level chosen by the function `Analyze`. If the analysis fails, it returns the backtracking level -1 and the algorithm terminates—the formula is true iff the losing player

Algorithm 4: Loss analysis for the existential player

Function `Analyze` (qlev,conflict)
begin
  $S_c \leftarrow \left\{ C \in \phi \mid \neg s_C^{\mathsf{qlev}-1} \in \mathsf{conflict} \right\}$
  $E \leftarrow \{l \mid l \in C, C \in S_c, l \text{ existential}, \mathsf{lv}(l) < \mathsf{qlev}\}$
  if $E = \emptyset$ **then return** $(-1, \cdot)$
  $\mathsf{btlev} \leftarrow \max(\{\mathsf{lv}(l) \mid l \in E\})$ `// btrack lev`
  $C_L \leftarrow \bigvee_{C \in S_c} \neg s_C$ `// learned clause`
  return (btlev, C_L)

is the universal player (ln. 12). One special case is treated at the beginning of the loop and this is when the quantification level is $n+1$ (ln. 7). This means that the existential player satisfied all the clauses (recall that $Q_n = \exists$), in which case we treat this situation as losing for the universal player.

4.1 Loss Analysis

Let us look more closely at how QESTO recovers from a loss of either of the players (lns. 11–14). Hence, we consider the situation where one of the conditions $\mathscr{C}_i$ becomes unsatisfiable under the choices of selected clauses made by the players so far. This situation is analyzed and two outcomes are computed: the quantification level btlev where the algorithm returns to, and, a strengthening (a *learned clause*) for the condition $\mathscr{C}_{\mathsf{btlev}}$ that will prevent the losing player from making the same mistake again. This is somewhat analogous to how SAT solvers perform conflict analysis.

To get the best out of the underlying SAT solver, we rely on the ability of the SAT solver to provide us with a *final conflict clause* for unsatisfiable calls. For a formula ψ and a conjunction of literals α—designated as *assumptions—final conflict clause* conflict is a clause that contains only variables appearing in the assumptions α and it holds that $\psi \models \mathsf{conflict}$. Which practically means that if $\psi \wedge \alpha$ is unsatisfiable, then $\psi \wedge \neg\mathsf{conflict}$ is also unsatisfiable and $\neg$conflict is a subset of α that is sufficient to arrive to unsatisfiability.

In our scenario, at level qlev, the assumptions are made up of the variables $s_C^{\mathsf{qlev}-1}$ and these signal to the SAT solver what choices were made so far. Hence, inspecting the clause conflict lets us infer with more precision which choices were responsible for the loss of the player at hand.[1]

Note that instead of the final conflict clause technique, we could have analyzed the corresponding *resolution refutation*—if the solver produces one [Zhang and Malik, 2003] (in fact only the leaves of the refutation are needed, similarly as in the *core extraction* techniques, which appear in a number of applications of SAT).

As the loss analysis is quite different for the two players, we separate it into two respective procedures. The analysis for the existential player is quite straightforward and is presented in Algorithm 4. The final conflict clause conflict gives us a set of selected clauses S_c at level $\mathsf{qlev}-1$ that lead to the

[1] This approach also enables us to use *incremental SAT* [Eén and Sörensson, 2003b] so that only the assumptions need to change when the same level is reached again.

Algorithm 5: Loss analysis for the universal player

```
1 Function Analyze (qlev,conflict)
2 begin
3     S_c ← { C ∈ φ | s_C^{qlev−1} ∈ conflict }
4     S'_c ← {C ∈ S_c | C not satisfied by an exist. lit}
5     if S'_C = ∅ then return (−1, ·)
6     btlev ← max ({ k | s_C^k = 0, C ∈ S'_c, k universal })
      R ← {C ∈ S_c | C sat. by an ex. lit at lev. > btlev}
7     return (btlev, ⋁_{C∈(S_c \ R)} s_C^{btlev})
```

unsatisfiability of $\mathscr{C}_{\mathsf{qlev}}$. To rectify this, the existential player must ensure that at least one of these clauses is *deselected*, the next time we get to the level qlev. For the existential player to deselect one of the clauses at a level $l < \mathsf{qlev}$ there must be an existential literal at that level. Otherwise, the existential player cannot influence the state of the clause. Hence, the backtrack level is calculated as the maximum over the levels of existential literals in the clauses S_c that are at a lower level than qlev. If there are no such literals, then the existential player cannot rectify the loss and the formula is solved (it is false). The learned clause is constructed so that it forces the deselection of at least one clause from the problematic set S_c.

The analysis for the universal player is inverse (Algorithm 5). At the time of a loss, there is some set of clauses S_c that are deselected and are causing the universal player to lose at the level qlev. If a clause is deselected by the existential player, the universal player cannot do anything about it. So we shift our attention to clauses $S'_l \subseteq S_c$ that are deselected by the universal player, i.e., satisfied by a universal literal. If there are no such clauses, the universal player cannot rectify the loss and therefore the formula is true. Otherwise, the backtrack level is computed as the maximum over the levels where the clauses in S'_c were deselected. Now we could define the learned clause as $\bigvee_{C \in S_c} s_C^{\mathsf{btlev}}$, but we can do better if we realize that the existential player can always repeat any choices that he has made *after* btlev. Hence, if the existential player was able to deselect a clause after btlev, he can do so again next time and therefore we remove these clauses from the learned clause.

4.2 Comparison to Other Approaches

In the context of a CNF formula ψ, a *maximal satisfiable set* (MSS) is a set of clauses $\psi' \subseteq \psi$ that is satisfiable and adding any clause from ψ to it makes it unsatisfiable. A complement $\psi \setminus \psi'$ of such set is called a *minimal correction set (MCS)*. Conversely, *minimal unsatisfiable set (MUS)* is a set $\psi' \subseteq \psi$ that is unsatisfiable and removing any of its clauses makes it satisfiable. It is well known that the set of all MUSes is the set of all minimal hitting sets of MCSes (and also the other way around). This duality of hitting sets is useful for calculating all MUSes or MUSes with certain properties, which has been exploited in a number of works [Liffiton and Sakallah, 2008; Previti and Marques-Silva, 2013]. Moreover, algorithms based on iteratively analyzing implicitly represented hitting sets find a growing number of applications [Karp, 2010; Chandrasekaran *et al.*, 2011; Davies and Bacchus, 2011; Stern *et al.*, 2012; Moreno-Centeno and Karp, 2013; Davies and Bacchus, 2013a; 2013b; Slaney, 2014]. The two-level approach we propose explicitly links this paradigm to QBF solving. Indeed, solving a two-level QBF ($\forall\exists$) can be phrased as a search for an MUS comprising of the clauses containing only the existential part. The universal part, imposes "rules" on which clauses may appear in an unsatisfiable set. Indeed, any clauses C_1, C_2 that contain universal literals $y \in C_1$, $\neg y \in C_2$ are never considered together. Algorithm 1 then iterates over satisfiable subsets of the matrix ϕ and requires that a complement of the satisfiable set is selected next time, which is consistent with the idea that MUSes are hitting sets of MCSes. Note that our algorithm does not to go over *maximally* satisfiable sets but it only requires satisfiable sets. Similarly, we do not require a *minimal* unsatisfiable set, but simply any unsatisfiable set. In our implementation, however, we make sure that the decision polarities of the selection variables in the SAT solver are set so that the SAT solver tends to find larger satisfiable sets.

The two-level approach can also be related to the counterexample guided abstraction refinement algorithm AReQS [Janota and Marques-Silva, 2011]. Indeed, the *selector variables* in our approach (Algorithm 1) can be seen as the *Tseitin variables* introduced in AReQS. We should stress, however, that the generalization of AReQS–RAReQS [Janota *et al.*, 2012]–is already completely different from our generalization (Algorithm 2). In RAReQS the generalization is done by recursion and the algorithm introduces a number of fresh variables throughout its course. While QESTO and RAReQS are similar for the two-quantification-level case, they are vastly different in the general case. Unlike RAReQS, QESTO does not introduce fresh variables as the algorithm progresses—selector variables are introduced at the very beginning and their number is bound. In this sense, our proposed algorithm differentiates itself from other expansion-based algorithms that convert the given formula into a SAT problem [Benedetti, 2004; Biere, 2004]. Further, RAReQS may create an exponential number of solvers due to the recursive calls while in QESTO, the number of solvers is always linear in the number of quantification levels (and it is fixed).

In its skeleton, our algorithm is similar to the *conflict/solution-driven learning* algorithms [Giunchiglia *et al.*, 2001; Zhang and Malik, 2002]; in the sense that it starts from the outermost quantification level, progresses to the inwards quantification level and backtracks whenever needed. It is, however, very different in how the algorithm is constructed and how learning is performed. Indeed, the building blocks of QESTO are a SAT solver and selection variables, which let us navigate the search space. This approach lets us devise a learning procedure anchored in the SAT solver's ability to analyze unsatisfiability (the final conflict clause), rather than building a dedicated unit propagator and learner; this gives us a strong engineering advantage.

5 Experimental Evaluation

We have implemented a prototype of Algorithm 2 named Qesto. The prototype used for the evaluation was imple-

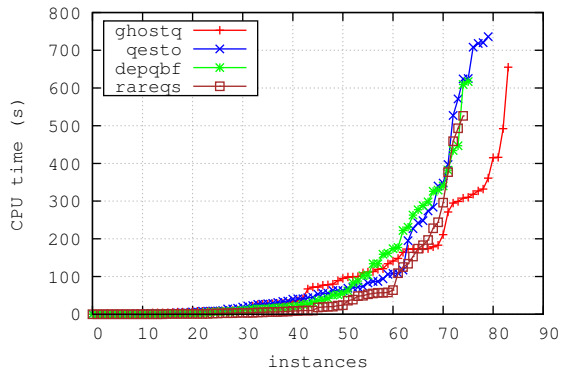

Figure 1: QBFLIB Benchmarks with 60 s cutoff

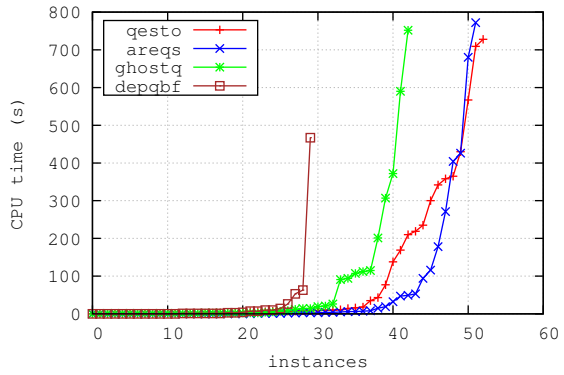

Figure 2: 2QBF Benchmarks

mented in C^{++} and the SAT solver minisat 2.2 [Eén and Sörensson, 2003a] was used. The experimental results were carried out on Linux machines with Intel Xeon 5160 3GHz processors and 4GB of memory. The time limit was set to 800 s and the memory limit to 2GB.

For the evaluation we used the *QBFLIB* benchmark suite used in the *2014 QBF Gallery*[2] and the benchmarks from the *2QBF track* of the *2010 QBF Evaluation*. All the instances were preprocessed by the preprocessor bloqqer [Biere *et al.*, 2011] and instances solved by the preprocessor were excluded from the evaluation. For comparison we have chosen a set of representative state-of-the-art QBF solvers. The conflict/solution-driven solver DepQBF [Lonsing and Biere, 2010]; the CEGAR-based solver RAReQS and its 2QBF version AReQS [Janota and Marques-Silva, 2011; Janota *et al.*, 2012]; and the non-CNF solver GhostQ [Klieber *et al.*, 2010]. Since GhostQ is a non-CNF solver, it was run on the unpreprocessed benchmarks as the solver's reverse engineering procedure benefits from their structure; it is also how it was run in the competition.

Table 1 summarizes the number of solved instances for the considered solvers. Figures 2 and 1 show cactus plots for the respective benchmark sets (cactus plots consist of points (x, y) meaning there are x instances solved in y sec). Since the QBFLIB cactus plot has a rather a long tail, the plot contains only those instances that were *not* solved within 60 seconds by all the solvers.

	ghostq	qesto	depqbf	(r)areqs
QBFLIB (276)	**137**	133	128	129
2QBF (65)	43	**53**	30	52

Table 1: Number of Solved Instances

GhostQ clearly dominates in the QBFLIB suite and it is followed by our prototype Qesto. There is a difference of only one solved instance between the solvers DepQBF and RAReQS. The success of the non-CNF solver GhostQ, confirms the well-known fact that representing QBF problems as CNF is harmful for solving [Ansótegui *et al.*, 2005; Klieber *et al.*, 2010; Goultiaeva and Bacchus, 2010] (we return to the topic in our future work discussion).

In the 2QBF suite our prototype Qesto solves the most instances. However, AReQS solves just 1 instance less, which comes as no surprise as the algorithms are closely related (see Section 4.2). Interestingly, GhostQ performs rather poorly on this set of benchmarks, which suggests that our prototype gives a higher degree of robustness when compared to others. More detailed presentation of the results can be found on the authors' website[3].

6 Summary and Future Work

This paper describes an algorithm QESTO to solve QBF in prenex form with a matrix in CNF. The algorithm is first presented in its two-level form ($\forall\exists$), which is shown to have a clear correspondence to implicit hitting sets, which appear in a number of applications of artificial intelligence. This two-level approach is also linked to the recent CEGAR-based two-level QBF algorithm AReQS [Janota and Marques-Silva, 2011]. The second half of the paper generalizes the two-level approach to QBFs with arbitrary number of quantification levels, giving rise to the QESTO algorithm. QESTO is an entirely novel approach to QBF solving and the experimental evaluation suggests that it is quite robust. It should also be noted that QESTO is quite advantageous from the engineering perspective since a SAT solver is utilized in a blackbox fashion. QESTO is also quite interesting from the theoretical perspective as it can be seen as a generalization of duality hitting set principle for PSPACE.

On the QBFLIB benchmarks QESTO was outperformed by the non-CNF solver GhostQ [Klieber *et al.*, 2010]. This suggests that it would be worthwhile investigating how QESTO could be applied to non-CNF formulas. We believe that this should be possible as this was already successfully done for conflicts/solution-driven learning algorithms [Zhang, 2006; Goultiaeva *et al.*, 2013]. Other improvements, such as *pure literals* or *variable dependencies*, are also considered for future work.

[2] http://www.kr.tuwien.ac.at/events/qbfgallery2013/benchmarks/eval2012r2.tar.7z

[3] http://sat.inesc-id.pt/%7Emikolas/sw/qesto/

References

[Ansótegui *et al.*, 2005] Carlos Ansótegui, Carla P. Gomes, and Bart Selman. The Achilles' heel of QBF. In *AAAI*, pages 275–281, 2005.

[Benedetti and Mangassarian, 2008] Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[Benedetti, 2004] Marco Benedetti. Evaluating QBFs via symbolic Skolemization. In *LPAR*, 2004.

[Biere *et al.*, 2011] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In *CADE*, 2011.

[Biere, 2004] Armin Biere. Resolve and expand. In *SAT*, 2004.

[Cadoli *et al.*, 1998] Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified Boolean formulae. In *AAAI 98*, pages 262–267, 1998.

[Chandrasekaran *et al.*, 2011] Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, and Santosh Vempala. Algorithms for implicit hitting set problems. In *SODA*, pages 614–629, 2011.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *CP*, pages 225–239, 2011.

[Davies and Bacchus, 2013a] Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MAXSAT. In *SAT*, pages 166–181, 2013.

[Davies and Bacchus, 2013b] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In *CP*, pages 247–262, 2013.

[Eén and Sörensson, 2003a] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, 2003.

[Eén and Sörensson, 2003b] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *ENTCS*, 89(4):543–560, 2003.

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified boolean logic satisfiability. In *IJCAI*, 2001.

[Giunchiglia *et al.*, 2009] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified boolean formulas. In *Handbook of Satisfiability*, pages 761–780. IOS Press, 2009.

[Goultiaeva and Bacchus, 2010] Alexandra Goultiaeva and Fahiem Bacchus. Exploiting QBF duality on a circuit representation. In *AAAI*, 2010.

[Goultiaeva *et al.*, 2011] Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *IJCAI*, pages 546–553. IJCAI/AAAI, 2011.

[Goultiaeva *et al.*, 2013] Alexandra Goultiaeva, Martina Seidl, and Armin Biere. Bridging the gap between dual propagation and CNF-based QBF solving. In *DATE*, pages 811–814, 2013.

[Janota and Marques-Silva, 2011] Mikoláš Janota and Joao Marques-Silva. Abstraction-based algorithm for 2QBF. In *SAT*, 2011.

[Janota *et al.*, 2012] Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. In *SAT*, 2012.

[Karp, 2010] Richard M. Karp. Implicit hitting set problems and multi-genome alignment. In *CPM*, page 151. Springer, 2010.

[Klieber *et al.*, 2010] William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *SAT*, 2010.

[Liffiton and Sakallah, 2008] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *JAR*, 40(1):1–33, 2008.

[Lonsing and Biere, 2010] Florian Lonsing and Armin Biere. DepQBF: A dependency-aware QBF solver. *JSAT*, 7(2-3):71–76, 2010.

[Moreno-Centeno and Karp, 2013] Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *OR*, 61(2):453–468, 2013.

[Narodytska *et al.*, 2014] Nina Narodytska, Alexander Legg, Fahiem Bacchus, Leonid Ryzhyk, and Adam Walker. Solving games without controllable predecessor. In *CAV*, pages 533–540. Springer, 2014.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Previti and Marques-Silva, 2013] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In *AAAI*, 2013.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[Rintanen, 1999] Jussi Rintanen. Improvements to the evaluation of quantified boolean formulae. In *IJCAI*, pages 1192–1197, 1999.

[Rintanen, 2007] Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *AAAI*, 2007.

[Slaney, 2014] John Slaney. Set-theoretic duality: A fundamental feature of combinatorial optimisation. In *ECAI*, pages 843–848, 2014.

[Stern *et al.*, 2012] Roni Tzvi Stern, Meir Kalech, Alexander Feldman, and Gregory M. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI*, 2012.

[Zhang and Malik, 2002] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *ICCAD*, 2002.

[Zhang and Malik, 2003] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885, 2003.

[Zhang, 2006] Lintao Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *AAAI*, 2006.