# Conflict Driven Learning in a Quantified Boolean Satisfiability Solver

Lintao Zhang
Department of Electrical Engineering
Princeton University
lintaoz@ee.princeton.edu

Sharad Malik
Department of Electrical Engineering
Princeton University
malik@princeton.edu

## ABSTRACT

Within the verification community, there has been a recent increase in interest in Quantified Boolean Formula evaluation (QBF) as many interesting sequential circuit verification problems can be formulated as QBF instances. A closely related research area to QBF is Boolean Satisfiability (SAT). Recent advances in SAT research have resulted in some very efficient SAT solvers. One of the critical techniques employed in these solvers is Conflict Driven Learning. In this paper, we adapt conflict driven learning for application in a QBF setting. We show that conflict driven learning can be regarded as a resolution process on the clauses. We prove that under certain conditions, tautology clauses obtained from resolution in QBF also obey the rules for implication and conflicts of regular (non-tautology) clauses; and therefore they can be treated as regular clauses and used in future search. We have implemented this idea in a new QBF solver called Quaffle and our initial experiments show that conflict driven learning can greatly speed up the solution process for most of the benchmarks we tested.

## 1. Introduction

Given a Quantified Boolean Formula, the question whether it is satisfiable (i.e. evaluates to 1) is called a Quantified Boolean Satisfiability (QBF) problem. Many practical problems ranging from AI planning [1] to sequential circuit verification [2] can be transformed to QBF instances. Recently, because of the wide industrial adoption of search based (in contrast to BDD based) sequential verification techniques, researchers in the EDA community are very interested in finding efficient QBF evaluation algorithms to make current search based sequential verification methods (e.g. bounded model checking [3]) complete.

Research on QBF solvers has been going on for some time. In [4], the authors present a resolution-based algorithm and prove that it is complete and sound. In [5], the authors present a decision procedure for QBF, which is similar to a resolution process. Both of these algorithms have the potential memory blow up problem encountered by most resolution based decision methods. Algorithms based on variations of the Davis Logemann Loveland (sometimes referred to as DPLL) procedure [6] such as [7] [8] [9] [10] [11] have also made a lot of progress. These methods use the classic DPLL algorithm without learning. Though methods based on traditional DPLL algorithm do not have the memory blow up problem, they require significant CPU cycles and are unable to handle practical sized problems as of now.

A closely related problem to QBF is the well-known Boolean Satisfiability problem (SAT). SAT can be regarded as a restricted form of QBF. In a SAT problem, only existential quantifiers are allowed. As SAT is very important both in theory as well as in practice, it has attracted significant research attention. Recent years have seen significant advancements in SAT research, resulting in some very efficient complete SAT solvers (e.g. GRASP [12], rel_sat [13], SATO [14], Chaff [15]). These solvers are also based on the DPLL algorithm. Most of these SAT solvers employ conflict driven learning and non-chronological backtracking techniques. Conflict driven learning utilize the knowledge learned from failures in certain search space to help prune search in future spaces. Experiments shows that conflict driven learning is very effective in pruning the search space for structured (in contrast to random) SAT instances. Because of the inherent similarity of search in QBF and SAT, the same idea of conflict driven learning should also help with structured QBF instances.

In this paper we present our work of incorporating conflict driven learning in a QBF solver. We assume that readers have some familiarity with state of the art SAT solvers, though every attempt will be made to explain the concepts used from that domain.

## 2. Problem Formulation

A QBF has the form

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

Where $\varphi$ is a propositional formula involving propositional variables $x_i$ (i=1…n). Each $Q_i$ is either an existential quantifier $\exists$ or a universal quantifier $\forall$. It is not a restriction to require $\varphi$ being in Conjunctive Normal Form (CNF), because there exist methods to translate any propositional logic formula into equivalent CNF (e.g. [17]). In the rest of the paper, we require that $\varphi$ be in CNF, i.e. it is constituted of a conjunction of *clauses*. Each clause is a disjunction of *literals*. A literal is the occurrence of a *variable* in either positive or negative *phase*. Because $\exists x \exists y \, \varphi = \exists y \exists x \, \varphi$ and $\forall x \forall y \, \varphi = \forall y \forall x \, \varphi$, we can

always group the quantified variables into disjoint sets where each set consists of adjacent variables with the same type of quantifier. In the rest of the paper, we will assume QBFs of the form:

$$Q_1X_{1.....}Q_nX_n \ C_1....C_m \qquad (2)$$

Here $C_j$'s are clauses, $X_i$'s are mutually disjoint sets of variables. Each variable in the formula must belong to one of these sets. We will call the variables existential or universal according to the quantifier of their respective quantifier sets. Also, each variable has a *quantification level* associated with it. The variables belonging to the outermost quantification set have quantification level 1, and so on. In the following, we will call the formula in form (2) a QBF in CNF.

Our framework is based on the DPLL procedure, which is a branch and search procedure on the variables. Therefore, in the rest of the paper, many of the statements will have the implicit "with regard to the current assignment of variable values" as a suffix. For example, when we say "the clause is conflicting", we mean that "the clause is conflicting with regard to current variable assignments". We will omit this suffix for concise presentation when no confusion can result. The value assignment to the variables may be a partial assignment. Some variables may not be assigned a value yet. We will call variables (and their corresponding literals) that have not been assigned *free*. Moreover, the DPLL algorithm is a branch procedure. Each branch has a *decision level* associated with it. The first branch variable has decision level 1, and so on. All of the variables implied by a decision variable will assume the same decision level as the decision variable. In the rest of the paper, we may use terms like "in the current branch". This has the same meaning as "in the partial variable assignment resulting from the implications of the current branching variables' assignments". All current QBF algorithms based on DPLL require that the branch order obeys the quantification order. A variable can be chosen as a branch variable if and only if all variables that has smaller quantification levels have already been assigned a value. In this paper, we assume that the quantification order is obeyed.

A *tautology clause* contains both a literal and its complement. We can assume that the initial input to the QBF solver has no tautology clauses. If it has, we can simply delete the tautology clauses without changing the propositional formula. A clause is *conflicting* if all of its existential literals evaluate to 0, and none of its universal literals evaluates to 1. A clause is *satisfied* if at least one of its literals evaluates to 1. A clause that is not satisfied is *unsatisfied*. If an unsatisfied clause has only one existential variable free, and all the free universal literals have quantification levels higher than the level of this free variable, then the clause is called a *unit clause*, and the free existential literal is called the *unit literal*.

If a clause is not a tautology, then it satisfies the following properties (assuming the quantification order is obeyed during branching). If it is a conflicting clause, then the current branch is unsatisfiable [7]. We will call this property the *Conflict Rule for Non-Tautology Clauses*. If it is a unit clause, the unit literal of the clause must evaluate to 1 in order to make the formula satisfiable in the current branch [7]. We will call this property the *Implication Rule for Non-Tautology Clauses*. The process of assigning all unit literals with value 1 is called *unit propagation*. When a variable is forced to assume a value by unit propagation, this variable is said to be *implied*. The clause that implies it is called the *antecedent* of the variable. Notice that only existential variables can be implied.

The basic framework of our algorithm for QBF solving is described in Figure 1. It differs from a SAT solving process (e.g. [12]) in some aspects. First, the branch procedure needs to obey the quantification order. Second, the deduction procedure uses the implication rule described earlier instead of the unit literal rule for SAT. Another difference of the procedure from DPLL for SAT is the case when deduction finds that the current branch is satisfiable. In SAT, it will immediately return with the solution found. In a QBF solver, we need to make sure both branches of a universal variable lead to a satisfiable solution. Therefore, the solver needs to backtrack and continue search.

The naïve DPLL algorithm will backtrack chronologically. Each decision variable will have a flag associated with it. The flag's value is either *flipped* or *unflipped*. When a decision is made, the flag will assume the initial value of *unflipped.* Function `analyze_conflict()` will find the unflipped *existential* decision variable with the highest decision level, flip

```
while (true) {
    decide_next_branch()                        //choose a branch variable
    while(true) {
        status = deduce();                      //unit propagation
        if (status == CONFLICT) {
            blevel = analyze_conflict();        //find out the reason for conflict
            if (blevel < 0)                     //even without branch, problem still unsat
                return UNSATISFIABLE;
            else backtrack(blevel);
        }
        else if (status == SATISFIABLE) {
            blevel = analyze_SAT()             //find out the reason for satisfactory
            if (blevel < 0)                     //even without branch, problem still sat
                return SATISFIABLE;
            else backtrack(blevel);
        }
        else break;
    }
}
```

**Fig. 1.** The top level DPLL algorithm for QBF evaluation.

it (i.e. make it assume the opposite phase, and mark it *flipped*), and return that decision level. Similarly, `analyze_SAT()` will find the unflipped *universal* decision variable with the highest decision level, flip it, and return the decision level. If no such decision variable exists, both will return -1.

The main topic of this paper is to improve the `analyze_conflict()` routine to make it smarter by determining the reasons for a conflict, and recording the reasons as clauses. These clauses can be used to prune the search space by avoiding making the same mistakes in the future. Recording reasons from conflicts is called conflict driven learning.

# 3. Conflict Driven Learning by Resolution

Conflict Driven Learning in Boolean SAT was proposed in the 1990's by Silva and Sakallah [12] and Bayardo and Schrag [13]. Conflict driven learning is the primary reason for the practical success of contemporary SAT solvers. Practical implementation of conflict-driven learning has been presented in the form of bi-partitioning of the implication graph in prior research (e.g. [12] 16]).

## 3.1 Conflict Driven Learning in Boolean SAT

```
analyze conflict(){
  if (current_decision_level()==0)
    return -1;
  cl = find_conflicting_clause();
  while (!stop_criterion_met(cl)) {
    lit = choose_literal(cl);
    var = variable_of_literal( lit );
    ante = antecedent( var );
    cl = resolve(cl, ante, var);
  }
  add_clause_to_database(cl);
  back_dl = clause_asserting_level(cl);
  return back_dl;
}
```

**Fig. 2** Conflict Analysis by Resolution

In this section, we present an alternate equivalent formulation of the learning process in a Boolean SAT solver using *resolution*. Resolution is a process to generate a clause from two clauses analogous to the process of *consensus* in the logic optimization domain (e.g. [19]).

We now define the process of *resolution* of two clauses $C_1$ and $C_2$ with respect to a variable $a$. Suppose clause $C_1$ contains literals $\{l_1, l_2, \ldots l_m, a\}$, clause $C_2$ contains literals $\{l_{m+1}, l_{m+2}, \ldots l_n, a'\}$, here $l_i$ $(1 \le i \le n)$ are literals, and $a$ and $a'$ are two literals corresponding to the same variable but have different phases. The *resolvent* of clause $C_1$ and $C_2$ with respect to $a$ is a clause containing literals $\{l_1, l_2, \ldots l_m, l_{m+1} \ldots l_n\}$. Similar to the well-known consensus law (e.g.[19]), the resulting clause of resolution between two clauses is redundant with respect to the original clauses. Therefore, we can always generate clauses from original clause database by resolution and add the generated clause back to the clause database without changing the satisfiability of the original formula.

For a SAT solver, the procedure of `analyze_conflict()` is shown in Figure 2. At the beginning, the function checks whether the current decision level is already 0. If that is the case, the function will return -1, essentially saying that there is no way to resolve the conflict and the formula is unsatisfiable. Otherwise, the solver determines the cause of the conflict. Iteratively, the procedure resolves the conflicting clause with the antecedent clause of a variable that appears in the conflicting clause. Function `choose_literal()` will always choose an implied literal (instead of the decision variable) from the input clause. Function `resolve(cl1,cl2,var)` returns a clause that has all the literals appearing in *cl1* and *cl2* except for the literals corresponding to *var*. If the generated clause meets some predefined stopping criterion, the iteration will stop; otherwise the resolution process is carried on iteratively. The actual learning is performed by `add_clause_to_database()`. In current state-of-the-art SAT solvers, `choose_literal()` always chooses the literals in reverse chronological order, i.e. the literal implied last in the clause will be chosen first. The stopping criterion is that the resulting clause be an *asserting clause*. An asserting clause is a clause with all value 0 literals, and among them, only one literal is at the current decision level. After backtrack, the clause will be a unit clause and this literal will be forced to flip, thus bringing the search to a new space. For more details about learning in a Boolean SAT solver, readers are referred to [16]. We want to point out that the formulation we provide here is equivalent to the usual implication graph partition formulation. For example, if the stop criterion is to stop at the *first* asserting clause, then it is exactly the same as the FirstUIP scheme described in [16].

## 3.2 Long Distance Resolution

In this section, we will discuss resolution as we use it in the context of QBFs. To clarify, we want to point out that we are interested in resolution here only because the learning process needs to use resolution to generate valid learned clauses. We do not imply that our solver framework is resolution based like the algorithms in [4] [5]. Our framework is based on the DPLL procedure; therefore, it does not have the same memory blow-up problem encountered by most resolution-based algorithms. In our framework learned clauses can always be deleted without affecting the correctness of the algorithm.

To apply the same procedure of Figure 2 to QBF solvers, we need to introduce a concept called *long distance resolution*. Similar to the concept of distance between two cubes in logic optimization (e.g. [19]), the distance between two clauses is the number of times a literal appears in positive phase in one of the clauses and in negative phase in the other clause. Note that in the SAT case, because one of the input clauses to `resolve()` is a conflicting clause (i.e. all literals evaluate to 0), and the other is an antecedent of the input variable (i.e. all but one literal evaluate to 0), the distance between these two clauses is always 1.

In QBF, the situation is different. Consider two clauses of a formula (there are other clauses in the QBF, but we only show two of them that we are most interested in):

$$(a_{(1)} + b_{(3)} + x_{(4)} + c_{(5)})\ (a_{(1)} + b'_{(3)} + x_{(4)}' + d_{(5)})$$

*a, b, c,* and *d* are existential variables, *x* is a universal variable. The numbers in the subscript are the quantification levels for the corresponding variables. Suppose initially, c and d are both implied to be 0 at decision level 5, and all the other literals are free. At decision level 6, suppose we make a branch *a=0*. By the implication rule, the first clause is unit because *x* has a higher quantification level than *b,* and *c* is already 0. Therefore, *b* is implied to be 1. This implication results in the second clause to become conflicting (because *a, b'*, and *d* are all 0). The function `analyze_conflict()` calls `resolve()` in Figure 2. The result of the resolution of clause 2 with the antecedent of *b* (i.e. the first clause) gives

$$(a_{(1)} + x_{(4)} + x_{(4)}' + c_{(5)} + d_{(5)}).$$

This clause is a tautology clause because we have both *x* and *x'* as literals in it. This is not a surprise because the consensus of two cubes with distance larger than 1 is an empty cube, and correspondingly the resolution of two clauses with distance greater than 1 is a tautological clause. However, in the following we will prove that we can regard such a resolvent clause between two clauses with distances greater than 1 as a regular (i.e. non-tautology) clause, and this clause will obey the same unit implication rule and conflict rule as regular clauses. In the previous example, because *a* has a lower quantification level than *x*; and *c* and *d* are assigned 0 at decision level 5, this last clause is a unit clause at decision level 5. Therefore, *a* should be implied at decision level 5 with value 1. Note that this does not follow immediately, as this clause is a tautology clause. The proof for the implication rule for non-tautology clauses in [7] will not apply here.

We will briefly outline our proof strategy in plain language first, and a more rigorous proof will follow. Clauses in a QBF solver have two functions. The first function is to provide non-conflicting implications, and thus lead the search to a new space, the second is to show conflicts, and thus declare that a certain search space has no solution. Therefore, to prove that clauses (regardless of tautology or not) generated from resolution can be regarded as regular clauses, we only need to prove that they also can be used to generate implications and conflicts by the same implication rule and conflict rule. Therefore, our main effort is to prove that if both of the two input clauses to the function `resolve()` in Figure 2 obey these two rules, then the output clause will also obey them (even if the output clause is a tautology clause). As we know, initially the entire clause database consists of non-tautology clauses, and they obey both rules. Therefore, we can use induction to prove that any clauses generated from the routine `analyze_conflict()` will obey these two rules, and can be regarded as regular clauses for implication and conflict generation in future search.

A note on the language used: In the following, we will not define all the terminology used due to space limitations. Sometimes we may use language informally if it does not cause confusion to improve readability. For example, we may say a rule holds for a clause, or a clause follows a rule, or a clause satisfies a rule etc. They all mean the same thing.

To prove that long distance resolution is feasible, we need to redefine some terms and formalize some assumptions.

**Definition 1**. Similar to the notion of distance between cubes or clauses, the distance between two sets of literals *S1* and *S2* is the number of literals *l* such that $l \in S1$, $l' \in S2$, We use *D(S1, S2)* to denote the distance between *S1* and *S2*.

For a clause C in a QBF in the form of (2), we use notation *E(C)* for the set of existential literals, and *U(C)* for the set of universal literals. We use *a, b, c…* (letters at the start of the alphabet) to denote existential literals, *x, y, z…* (letters at the end of the alphabet) to denote universal literals, and *V(a), V(b)…* to denote the value of the literals. If literal *a* is free, *V(a) = X*. We use *L(a), L(b)…* to denote the quantification levels of the variables corresponding to the literals.

**Definition 2**. Implication Rule: For a QBF F, if under a certain variable assignment, a clause *C* has literal *a* s.t.

1. $a \in E(C)$, $V(a) = X$. *If* $b \in E(C)$ *and* $b \neq a$, *then* $V(b)=0$.
2. $\forall x \in U(C)$, $V(x) \neq 1$. *If* $V(x) = X$, *then* $L(x) > L(a)$

Then the formula F can be satisfied in the branch only if we assign $V(a)=1$.

**Definition 3**. Conflict Rule: For a QBF F, if under a certain variable assignment, there exists a clause *C*, s.t. $\forall a \in E(C), V(a) = 0$, and $\forall x \in U(C), V(x) \neq 1$, then F cannot be satisfied in the branch.

Note that in these two rules, we do not require the clause to be non-tautology. Therefore, these two rules are definitions instead of lemmas because they are not shown to be valid yet. In the following, we will prove that they actually hold for all the clauses that can appear in the database if we follow the procedure depicted in Figure 2.

**Lemma 1**. If for clause *C*, $\forall x \in U(C), x' \notin U(C)$, then the clause *C* follows both the Implication Rule and the Conflict Rule.

Proof: If $\forall a \in E(C), a' \notin E(C)$, then the clause C is not a tautology clause. This is essentially Lemma 2.6 of [7]. Otherwise, it is impossible for this clause to satisfy the conditions of the Implication Rule and Conflict rule, so they obviously hold.

**Lemma 2**. If clause *C1* and *C2* both follow the Conflict and Implication Rule, and they satisfy:

1. *D(E(C1), E(C2)) = 1. Let the distance 1 literal be a. i.e.* $a \in E(C1), a' \in E(C2)$,
2. *For any x, if* $x \in U(Ci)$ and $x' \in U(Cj)$, $i,j \in \{1,2\}$ *then* $L(x) > L(a)$

Then, the resolvent clause C with $E(C) = (E(C1) \backslash a) \cup (E(C2) \backslash a')$ and $U(C) = U(C1) \cup U(C2)$ also obeys these two rules.

Proof: To simplify the notation, define $E'(C1) = E(C1) \backslash a$, $E'(C2) = E(C2) \backslash a'$. We will prove each rule separately.

a) Clause C obeys the Conflict Rule.

Because of condition 1, $D(E'(C1), E'(C2)) = 0$. Therefore, there may have some variable assignments that satisfy both of

the conditions to the Conflict Rule. i.e. the assignment can make $\forall b \in E(C), V(b) = 0, \forall x \in U(C), V(x) \neq 1$. We will prove that if that is the case, the QBF is unsatisfiable in this branch. There are three cases:

1. If $V(a) = 0$, $C1$ becomes the conflicting clause.
2. If $V(a) = 1$, $C2$ becomes the conflicting clause.
3. If $V(a)=X$, we will demonstrate an assignment of universal variables that leads to a conflict. For any $x \in U(C)$, $L(x) < L(a)$, we assign the variable corresponding to $x$ s.t. $V(x) = 0$. This can be done because condition 2 guarantees that if $L(x) < L(a)$, and $x \in U(C)$ then $x' \notin U(C)$. Then, both clause $C1$ and $C2$ satisfy the condition of Implication Rule. Clause $C1$ requires $V(a) = 1$, while clause $C2$ requires $V(a') = 1$. This leads to conflict.

Therefore, the Conflict Rule holds.

b) Clause C obeys the Implication Rule.

Suppose there exists $l \in E(C), \forall b \in E(C), b \neq l, V(b)=0$ and $\forall x \in U(C), V(x) \neq 1$. If $V(x) = X$, then $L(x) > L(l)$. We want to prove that such a situation implies that the branch can be satisfiable only if $V(l)=1$.

Because $l$ has the smaller quantification level than that of any of the free universal literals appearing in $C$, it must be assigned a value before any of them. Therefore, if the assignment makes $V(l) = 0$, then the clause $C$ satisfies the condition of the Conflict Rule, thus by the first part of the proof, it is a conflict. Therefore, F can be satisfied in the branch only if $V(l)=1$.

Therefore, the Implication Rule also holds.

**Theorem.** The clause generated by the procedure depicted in Figure 2 obeys both Implication Rule and Conflict Rule.

Proof: We will prove this by induction on the resolution depth of the resulting clause. We define resolution depth recursively. The resolution depth of the clause generated from the `resolve()` function is the maximum resolution depth of the two input clauses plus 1. Initially, all of the original clauses have resolution depth 0.

Induction Basis: We require that no input clause be a tautology. Therefore, if a clause has resolution depth 0, it satisfies both rules by Lemma 1.

Induction Hypothesis: The theorem holds for resolution depth smaller or equal than n.

Induction Step: Clause $C$ with resolution depth of n+1 is obtained by calling `resolve()` on two clauses $C1$ and $C2$. Both $C1$ and $C2$ have resolution depth less than or equal to n. Therefore, both $C1$ and $C2$ follow the Implication Rule and the Conflict Rule. We will prove that these two clauses satisfy both condition 1 and 2 of Lemma 2, and thus the theorem holds.

Suppose $C2$ is the antecedent of the resolution variable *var*, the corresponding literal for *var* is $a$. Then obviously, $V(a)=1$, $\forall l \in E(C2), l \neq a, V(l)=0$. $C1$ is obtained from a conflicting clause by some resolution steps. All the other clauses involved in these resolution steps are unit clauses, and the unit literals are removed during the resolution process. Therefore, $\forall l \in E(C1), V(l)=0$. Therefore, $D(E(C1), E(C2))=1$. Thus condition 1 of the lemma is satisfied.

Suppose there is a literal $x \in U(C2)$, Then if $x' \in U(C2), L(x) > L(a)$ because otherwise $a$ will not be implied by this clause. If $x' \in U(C1)$, we also have $L(x) > L(a)$ because otherwise, for $a$ to be implied in $C2$, we need $V(x)=0$. But then $V(x')=1$, therefore the clause that has $x'$ in it will neither be a unit clause nor a conflicting clause, so this literal should not appear in $C1$. Thus condition 2 of the lemma is satisfied. ■

Notice in our proof, we only require that $D(E(C1), E(C2)) = 1$. We do not have any requirements for the distance between the universal literals. Therefore, the distance between these two clauses may be larger than 1. That is the reason why we call this *long distance resolution*. Our theorem proves that we can just regard the results from the resolution process as regular non-tautology clauses for purposes of the Implication and Conflict Rules, even though some of these resolvents may contain universal literals in both phases.

In [4], the authors proposed a concept called *Q-resolution*. Q-resolution omits each universal variable in a resulting clause if no existential variable in the clause has higher quantification level than the universal variable. Therefore, in Q-resolution, the resulting clause may contain fewer variables than an ordinary resolvent from the same two clauses. The authors prove that Q-resolution is valid in QBF, i.e. the result from a Q-resolution is redundant and can be added to the original clause database. Our proof for the main theorem is valid even if "Q-resolution" is used instead of regular resolution.

In our proof for the Implication Rule, we used the assumption that a universal variable can be assigned a value only if all variables that have quantification level less that it have already been assigned. The Monotone Literal Rule [7] (sometimes called pure literal rule, which states that a variable can be assigned a value if all of its occurrences in the unsatisfied clauses are in the same phase. Please refer to [7] for details about this rule) actually invalidates this assumption. However, the Monotone Literal Rule will not invalidate our conclusion because if applied, it will only make the universal literals evaluate to 0. Therefore, the Implication Rule still holds even when the Monotone Literal Rule is applied. Because the proof for the Conflict Rule does not depend on the above mentioned assumption, the rule obviously holds.

## 3.3 Stop Criterion for QBF

The stop criterion will make sure that the resulting clause from the resolution will generate a clause that can really resolve the current conflict, and bring the search to a new space.

In a QBF solver, the function `choose_literal()` will still only choose an implied literal because decision variables do not have an antecedent. Therefore, it automatically makes sure that all the chosen resolved variables will be existential.

The stop criterion for QBF is different from SAT. The resolution process should stop if the generated clause satisfies the following conditions:

1. Among all its existential variables, one and only one of them has the highest decision level (which may not be the current decision level). Suppose this variable is V.
2. V is in a decision level with an existential variable as the decision variable.
3. All universal literals with quantification level smaller than V's are assigned 0 before decision level of V's.

The first and third conditions make sure that this clause is indeed an asserting clause (i.e. unit after backtracking). The second condition makes sure that we will undo at least one existential decision because undoing universal decisions only can never really resolve a conflict. An exception to this stop criterion is that if all existential literals in the resulting clause are at decision level 0, or if there are no existential literals in the resulting clause, then the solver should stop and immediately state that the problem is unsatisfiable.

### 3.4 Implementing the QBF solver Quaffle

We have implemented the above-mentioned idea in a new QBF solver called Quaffle. Quaffle is a loose acronym for Quantified Formulae Evaluator with Learning. Quaffle is implemented in C++. We have implemented both the naïve chronological backtracking scheme as well as the new conflict driven learning scheme in the place of `analyze_conflict()`, as shown in Figure 2. In the learning case, we use the decision strategy VSIDS ([15]), which has been shown to be quite successful for Boolean SAT solvers. In the naïve case, we still used VSIDS, but disabled the decaying mechanism, which makes sense only with learned clauses, thus just choosing the unassigned literal that occurs the most. In all the experiments in this paper, Quaffle uses the naïve chronological backtracking scheme in place of `analyze_SAT()`.Section 5 provides some additional information on further developments on this.

The learned clauses from conflicts can be deleted in the same manner as in Boolean SAT solvers. It is easy to implement either relevance based or clause length based clause deletion. In the current implementation, because the benchmarks are not very big and the solver is largely runtime limited, we did not turn the clause deletion on in all the experimental runs. For larger benchmarks, certain clause deletion strategies are obviously needed. It is also possible to implement other techniques commonly used in SAT solvers such as random restart in our framework. However, a detailed discussion and experimentation for that is beyond the scope of this paper.

## 4. Experimental Results

In this section, we compare the performance of three versions of Quaffle. The first version has the learning turned on, which is denoted as Quaffle-CDL in the result table shown in Figure 3. The second version uses the same procedure as Quaffle-CDL, except that the learned clauses are deleted immediately. The result is that we essentially disabled learning but enabled the solver to perform non-chronological backtracking. This is equivalent to the concept of backjumping in [11], and we denote it Quaffle-BJ in the table. The third version, denoted Quaffle-naïve in the table, is using the naïve version of `analyze_conflict()`. We also compare our solver's performance with some of the existing state-of-the-art QBF solvers. We chose two of the most efficient solvers for the test suite we use. For a comprehensive comparison of how other state-of-the-art QBF solvers fare on these benchmarks, we refer the readers to [18]. QuBE-BJ is the backjumping [11] version of QuBE [10], while QuBE-BT is the version with simple backtracking scheme. Decide [8] is the solver by J. Rintanen.

The benchmark suite was obtained from J. Rintanen's home page [20]. We are unable to provide details on the benchmarks (e.g. number of variables, number of clauses etc.) because of space limitations. These, as well as additional benchmarks are available on a website for this paper [21]. The benchmarks shown here consist of some random QBF instances as well as some instances generated from real applications. The tests were conducted on a Dell PIII 933 machine with 1G memory. The timeout limit is 1800 seconds for each instance. We omitted the instances for which all the solvers timeout.

Currently, Quaffle does not have any of the advanced techniques employed in other state-of-the-art QBF solvers such as pure literal rule [7], trivial truth [7], inversion of quantifiers [8], implicit unrolling [9], connected component detection [8] etc. Therefore, it is not surprising that the naïve version of Quaffle is not very competitive compared with others. The Quaffle-BJ version fares much better than Quaffle-naïve, because of non-chronological backjumping. In comparison, it is doing better in this particular benchmark suite than QuBE-BJ. We suspect that is so because our branch heuristic VSIDS is better suited for these examples. Notice that in most of the benchmarks, Quaffle-CDL is much better than Quaffle-BJ. This shows the effectiveness of conflict driven learning in a QBF setting compared with back jumping only. The only exceptions are four of the TOILET test cases. We suspect that is due to noise introduced in learning that brings the search to some area that is hard to get out of. Random restarts and other techniques that can bring search out of the "valley area" may help in these cases.

We want to point out that the effectiveness of conflict driven learning depends on how many conflicts are encountered in the searching process. In some of the benchmarks (e.g. CHAIN), almost all of the terminal leaves of the search are satisfying leaves (i.e. in Figure 1, the status is SAT). In that case, learned clauses do not contribute much compared with just back jumping. We would like to show some representative data for the search tree size, number of implications, number of conflicts etc. for each solver. However, we do not have access to the source code of other solvers, and their print outs after each run are not quite informative. All three solvers use different implication rules, thus making direct comparison of number of branches misleading. For example, decide [8] uses a look ahead technique for each branch, this will reduce the number of decisions, but will increase the time need for each decision. Therefore, we can only show the run time for each of the

| Benchmark Name | | Quaffle -CDL | Quaffle -BJ | Quaffle -naïve | QuBE -BT | QuBE -BJ | Decide | Benchmark Name | | Quaffle -CDL | Quaffle -BJ | Quaffle -naïve | QuBE -BT | QuBE -BJ | Decide |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BLOCKS3i.4.4 | F | 0.07 | 0.07 | - | - | - | 0.05 | impl20 | T | 15.82 | - | - | - | - | - |
| BLOCKS3i.5.3 | F | 29.92 | 128.42 | - | - | - | 31.89 | logn...A0 | F | 0 | 0 | 0 | 0 | 0 | 0.01 |
| BLOCKS3i.5.4 | T | 2.91 | 30.75 | - | - | - | 6.59 | logn...A1 | F | 2.23 | 67.47 | - | - | 7.28 | 0.47 |
| BLOCKS3ii.4.3 | F | 0.05 | 0.07 | - | - | 5.02 | 0.03 | logn...A2 | T | 127.4 | - | - | - | - | 28.28 |
| BLOCKS3ii.5.2 | F | 0.13 | 0.82 | - | - | 82.19 | 0.07 | logn...B0 | F | 0.01 | 0.01 | 0.01 | 0 | 0 | 0.03 |
| BLOCKS3ii.5.3 | T | 0.33 | 2 | - | - | - | 1.5 | logn...B1 | F | 8.47 | 342.55 | - | - | 37.89 | 0.61 |
| BLOCKS3iii.4 | F | 0.03 | 0.05 | - | - | 1.62 | 0.01 | logn...B2 | F | 767.94 | - | - | - | - | 1.88 |
| BLOCKS3iii.5 | T | 0.28 | 0.72 | - | - | - | 0.26 | R...3...50_0.T | T | 1.23 | 3.15 | - | 0 | 0 | 0.03 |
| BLOCKS4i.6.4 | F | 254.11 | - | - | - | - | 5.35 | R...3...50_1.F | F | 0.02 | 2.52 | - | 0.03 | 0.01 | 0.47 |
| BLOCKS4ii.6.3 | F | 400.99 | - | - | - | - | 4.53 | R...3...50_2.T | T | 0.83 | 1.11 | 738.7 | 0 | 0 | 0.05 |
| BLOCKS4ii.7.2 | F | - | - | - | - | - | 9.15 | R...3...50_3.T | T | 1.07 | 1.65 | 521.98 | 0.11 | 0 | 0.06 |
| BLOCKS4ii.7.3 | T | - | - | - | - | - | 731.24 | R...3...50_4.T | T | 1.44 | 2.74 | - | 0.06 | 0.01 | 0.1 |
| BLOCKS4iii.6 | F | 40.27 | - | - | - | - | 2.92 | R...3...50_5.T | T | 0.97 | 21.42 | 71.41 | 0.01 | 0 | 0.07 |
| BLOCKS4iii.7 | T | - | - | - | - | - | 414.76 | R...3...50_6.F | F | 1.53 | 11.99 | - | 0.07 | 0.03 | 20.52 |
| CHAIN12v.13 | T | 0.31 | 0.32 | - | 0.36 | 0.34 | 0.41 | R...3...50_7.F | F | 0.6 | 7.88 | - | 0.05 | 0.02 | 1.91 |
| CHAIN13v.14 | T | 0.69 | 0.69 | - | 0.84 | 0.8 | 0.79 | R...3...50_8.F | F | 0.28 | 1.33 | - | 0.22 | 0.06 | 0.32 |
| CHAIN14v.15 | T | 1.47 | 1.49 | - | 2.45 | 2.27 | 1.6 | R...3...50_9.T | T | 0.87 | 1.03 | 40.99 | 0.05 | 0.01 | 0.25 |
| CHAIN15v.16 | T | 3.19 | 3.25 | - | 4.57 | 5.17 | 3.25 | R...7...60_0.F | F | 0.13 | 2.19 | 479.68 | 0.44 | 0.02 | 0.01 |
| CHAIN16v.17 | T | 6.9 | 7.03 | - | 15.53 | 13.38 | 6.7 | R...7...60_1.T | T | 0.23 | 0.52 | 0.55 | 0.07 | 0.01 | 0.06 |
| CHAIN17v.18 | T | 15.27 | 15.14 | - | 34.89 | 43.44 | 14.48 | R...7...60_2.T | T | 1.28 | 0.65 | 423.71 | 0 | 0 | 0.05 |
| CHAIN18v.19 | T | 32.21 | 33.02 | - | 97.73 | 100.06 | 31.21 | R...7...60_3.T | T | 0.33 | 0.51 | 36.83 | 0.75 | 0.02 | 0.15 |
| CHAIN19v.20 | T | 74.04 | 75.09 | - | 234.65 | 249.04 | 61.08 | R...7...60_4.T | T | 13.44 | 51.64 | - | 0.76 | 0.01 | 0.04 |
| CHAIN20v.21 | T | 152.88 | 166.98 | - | 527.92 | 650.44 | 130.71 | R...7...60_5.F | F | 1.32 | 8.32 | - | 0.64 | 0.01 | 27.12 |
| CHAIN21v.22 | T | 340.29 | 362.24 | - | 1271 | 1462.59 | 272.58 | R...7...60_6.T | T | 0.51 | 1.79 | 150.08 | 0.19 | 0.03 | 0.83 |
| CHAIN22v.23 | T | 741 | 846.08 | - | - | - | 569.65 | R...7...60_7.T | T | 2.22 | 11.9 | - | 0.73 | 0.06 | 0.18 |
| CHAIN23v.24 | T | 1783.73 | 1790.14 | - | - | - | 1200.91 | R...7...60_8.F | F | 0 | 0 | 1.85 | 0.02 | 0.01 | 0.08 |
| impl02 | T | 0 | 0 | 0 | 0 | 0 | 0 | R...7...60_9.T | T | 0.23 | 3.75 | - | 0.02 | 0 | 0.09 |
| impl04 | T | 0 | 0 | 0 | 0 | 0 | 0.01 | TOILET02.1.iv.3 | F | 0 | 0 | 0 | 0 | 0 | 0 |
| impl06 | T | 0.01 | 0.01 | 0.01 | 0 | 0.01 | 0.04 | TOILET02.1.iv.4 | T | 0 | 0 | 0 | 0 | 0 | 0 |
| impl08 | T | 0.01 | 0.04 | 0.07 | 0.01 | 0 | 0.29 | TOILET06.1.iv.11 | F | 40.15 | 7.46 | 757.08 | 25.68 | 5.4 | 10.45 |
| impl10 | T | 0.02 | 0.33 | 0.48 | ** | ** | 2.29 | TOILET06.1.iv.12 | T | 18.17 | 5.57 | 1633.03 | 12.36 | 1.48 | 0.1 |
| impl12 | T | 0.06 | 2.34 | 3.57 | ** | ** | 17.38 | TOILET07.1.iv.13 | F | - | 104.47 | - | 599.6 | 92.06 | 141.17 |
| impl14 | T | 0.25 | 18.19 | 26.41 | ** | ** | 130.64 | TOILET07.1.iv.14 | T | - | 74.77 | - | 265.3 | 23 | 0.22 |
| impl16 | T | 0.99 | 135.22 | 195.52 | ** | ** | 974.53 | TOILET10.1.iv.20 | T | - | | - | - | - | 1.31 |
| impl18 | T | 3.98 | 985.57 | 1445.76 | ** | ** | - | TOILET16.1.iv.32 | T | - | | - | - | - | 15.39 |

Figure 3. Run time (in seconds) comparison of Quaffle with other solvers
** Qube generated segmentation fault on these benchmarks
- Solver aborted after 1800 seconds

solvers, because that is the ultimate criterion in evaluating a solver.

From the experimental results we find that Quaffle is quite competitive compared with other state-of-the-art QBF solvers. It seems that conflict driven learning has great effect in pruning the search space, as shown by comparing the naïve version and CDL version of Quaffle. Quaffle with CDL outperforms Quaffle-BJ in most of the structured benchmarks, demonstrating that CDL is more effective than simple backjumping. Decide fares very well on this particular benchmark suite, mainly because the "inversion of quantifier" [8] and "implicit unrolling" [9] heuristic it employs seem to be very effective for these particular benchmark suites. Our focus in this paper was to study conflict driven learning for QBF, and thus our experiments are currently focused on evaluating the gains provided by this capability.

## 5. Related Work

A closely related work is presented by E. Giunchiglia *et al.* recently in [11]. In that paper, the authors demonstrate how to add backjumping into their QBF solving process. Our work differs from their conflict directed backjumping in the way that we keep the knowledge from conflicts as learned clauses, and non-chronological backjumping is a direct consequence of the learned clauses much like in modern SAT solvers. As this learned clause is recorded in the database, it may be used for future pruning of the search space. In contrast, in [11] learning is not possible. On the other hand, the paper has the concept of solution directed backjumping, which can prune the search space when a satisfying leaf is encountered. In this work, we do not have a corresponding concept for that. We have independently developed new techniques to deal with this deficiency in [24].

Recently there has been a big resurgence in QBF research. We are now aware of two research efforts with contemporaneous publications. In particular, in [22], the authors propose to cache lemmas and models to help the search. The idea of cache lemmas is similar to the idea discussed here about conflict driven learning. In that work, even though the authors used resolution (more specifically, Q-resolution [4], a stronger version of regular resolution for QBF) to generate lemmas, they do not show that tautology clauses can be treated exactly the same as regular clauses, as shown by this work. In [23], the authors also proposed to incorporate conflict driven learning in their QuBE framework. However, their approach is not resolution based, thus needs some special treatment (i.e. pre-fix closed) for the variable assignments in order to construct reasons correctly for learning. These two papers also developed schemes for learning for satisfied leaves, denoted model caching in [22] and good solution learning in [23] respectively. Basically, they are improving the function `analyze_SAT()` discussed in this paper with more powerful schemes. We have also independently proposed our own version of the idea in [24], which expands the work presented here. For a more detailed comparison of these solvers, we refer readers to [21] for more experiment results.

## 6. Conclusion

In this paper we present our work on incorporating conflict driven learning into a Quantified Boolean Formulae evaluation framework. Conflict driven learning has been tremendously successful in Boolean SAT solvers. Our experiments show that it is also very promising in a QBF solver framework. To adapt the algorithm for the QBF case, we introduce long distance resolution and prove that the resulting clauses obey the rules of regular clauses. We implemented this algorithm in a new QBF solver called Quaffle. Experiments shows that conflict driven learning can greatly speed up the solving process.

## 7. References


[1] J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research, 10:323-352, 1999*

[2] M. Sheeran, S. Singh, G. Stälmark, Checking Safety Properties Using Induction and a SAT-Solver, in *Proc. of FMCAD, 2000*

[3] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs, In *Tools and Algorithms for the Analysis and Construction of Systems (TACAS), 1999*

[4] H. Kleine-Büning, M. Karpinski and A. Flögel. Resolution for quantified Boolean formulas. In *Information and Computation, 117(1):12-18, 1995*

[5] D. A. Plaisted, A. Biere and Y. Zhu. A Satisfiability Procedure for Quantified Boolean Formulae, to appear, *Discrete Applied Math.*

[6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. In *Communications of the ACM, 5:394-397, 1962*

[7] M. Cadoli, M. Schaerf, A. Giovanardi and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation, in *Highlights of Satisfiability Research in the Year 2000, IOS Press, 2000*

[8] J. Rintanen, Improvements to the Evaluation of Quantified Boolean Formulae, in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999*

[9] J. Rintanen, Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae, in *International Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), 2001*

[10] E. Giunchiglia, M. Narizzano and A. Tacchella,. Qube: a system for Deciding Quantified Boolean Formulas Satisfiability,. In *Proceedings of International Joint Conference on Automated Reasoning (IJCAR), 2001*

[11] E. Giunchiglia, M. Narizzano and A. Tacchella. Backjumping for Quangified Boolean Logic Satisfiability. In *Proceedings of International Joint Conf. on Artificial Intelligence (IJCAI), 2001*

[12] João P. Marques-Silva and Karem A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability, In *IEEE Transactions on Computers, vol. 48, 506-521, 1999*

[13] R. Bayard and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances, in *Proc. of the 14th Nat. (US) Conf. on Artificial Intelligence (AAAI), 1997*

[14] H. Zhang. SATO: An efficient propositional prover, In *Proc. of the International Conf. on Automated Deduction, 1997*

[15] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT Solver, In *Proc. of the Design Automation Conference (DAC), 2001*

[16] L. Zhang, C. Madigan, M. Moskewicz, S. Malik, Efficient Conflict Driven Learning in a Boolean Satisfiability Solver, in *Proc. of International Conf. on Computer Aided Design (ICCAD), 2001*

[17] D.A. Plaisted and S. Greenbaum. A Stucture-preserving Clause Form Translation, in *J. of Symbolic Computation, 2:293-304, 1986*

[18] E. Giunchiglia, M. Narizzano and A. Tacchella, On the effectiveness of backjumping and trivial truth in quantified boolean formulas satisfiability, in *IJCAR workshop on Theory and Application of Quantified Boolean Formulas, 2001*

[19] G. Hachtel and F. Somenzi, Logic Sysntheiss and Verification Algorithms, *Kluwer Academic Publishers*

[20] J. Rintanen's benchmark suites are available at http://ww.informatik.uni-freiburg.de/~rintanen/qbf.html

[21] More detailed and expanded experimental results are available at http://ee.princeton.edu/~chaff/Quaffle.php

[22] R. Letz, Lemma, Model Caching in Decision Procedures for Quantified Boolean Formulas, in *International Conf. on Automated Reasoning with Analytic Tableaux and Related Methods, 2002*

[23] E. Giunchiglia, M. Narizzano and A. Tacchella, Learning for Quantified Boolean Logic Satisfiability, in *Proc. of the 18th Nat. (US) Conf. on Artificial Intelligence (AAAI), 2002*

[24] L. Zhang, S. Malik, Towards a symmetric treatment of satisfaction and conflicts in Quantified Boolean Formula Evaluation, in *Proc. Eighth International Conference on Principles and Practice of Constraint Programming (CP2002), 2002*