Local and Global Complete Solution Learning Methods for QBF

Ian P. Gent[1] and Andrew G.D. Rowley[2]

[1] School of Computer Science, University of St Andrews, St. Andrews, Fife, UK; ipg@dcs.st-and.ac.uk
[2] Manchester Computing, Kilburn Building, Oxford Road, Manchester, UK; andrew.rowley@manchester.ac.uk

Abstract. Solvers for Quantified Boolean Formulae (QBF) use many analogues of technique from SAT. A significant amount of work has gone into extending conflict based techniques such as conflict learning to solution learning, which is irrelevant in SAT but can play a large role in success in QBF. Unfortunately, solution learning techniques have not been highly successful to date. We argue that one reason for this is that solution learning techniques have been 'incomplete'. That is, not all the information implied in a solution is learnt. We introduce two new techniques for learning as much as possible from solutions, and we call them complete methods. The two methods contrast in how long they keep information. One, Complete Local Solution Learning, discards solutions on backtracking past a previous existential variable. The other, Complete Global Solution Learning, keeps solutions indefinitely. Our detailed experimental analysis suggests that both can improve search over standard solution learning, while the local method seems to offer a more suitable tradeoff than global learning.

1 Introduction

Quantified Boolean Formulae (QBFs) can be seen as an extension of Satisfiability (SAT) with the addition of a prefix in which every variable is universally or existentially quantified. This increase in expressivity comes at a price; the decision problem for QBF is PSPACE-complete. This allows QBF to be used to solve many other PSPACE problems such as games, conditional planning problems and hardware verification.

There have been many advances in QBF solver technology. Many of the more recent algorithms are based on an extension to the DPLL [3] procedure for SAT. The first such example is presented in [2]. Many other SAT techniques have been extended and added to this procedure, starting with conflict and solution directed backjumping [6], in which pruning of the search tree is achieved through look-back techniques. This provided a great improvement in both run-time and branching rate. Learning was then introduced [5, 7, 10]. This had much less impact than occurred in SAT. While improvements were seen on some problems, the algorithms actually performed worse on others.

Recently, it has been shown that solution learning for quantified Boolean formulae is not very good in practice. In two separate papers [5, 10], the experimental analysis of solution and conflict learning show that their combination does not perform well in practice. Two other papers [7, 9] show that conflict learning for QBF used independently performs quite well. This implies that the detrimental effect of the combination learning solver was likely to be due to the solution learning part of the solver. This effect is bad enough to make the combination of conflict and solution learning perform poorly.

There are at least two reasons why solution learning might not perform well in practice. Firstly, the level of solution learning performed was unrestricted learning. This means that when a solution is learned, it is kept for the remainder of the search. An alternative approach would be to perform some sort of bounded learning. This was suggested in [5] but was never implemented. A second reason is that the solutions learned may not be the most useful in practice. As was stated in [10], there are many possible sets which can be learned on discovery of a solution. There may be an exponential number of such solutions which would result in an exponential space requirement if all solutions are learned using this technique.

We introduce a method of solution learning which addresses both of the aforementioned issues. This technique will allow the learning of all solutions without the exponential space requirement. Furthermore, this technique will automatically throw away learned information when it is no longer of any use. This will be a requirement of the information as it will become invalid under certain conditions. A second technique will also be introduced which, in extension to the first technique, will allow the storage of the learned information for the duration of the search. This will allow other bounding algorithms to be applied to the learning.

First, a proper introduction to solution learning in QBF will be given. This will be followed by a description of the first new technique. The extension to this technique will then be described. Finally, an experimental evaluation of the techniques will be given. This will show that the new technique performs better than the standard solution learning technique on many instances.

2 Background

We provide some basic details of QBF, but refer the reader elsewhere for a more detailed introduction [2]. A QBF is of the form QC where Q is a sequence of quantifiers $Q_1v_1...Q_nv_n$, where each Q_i quantifies ($\exists$ or $\forall$) a variable v_i and each variable occurs exactly once in the sequence Q. C is a Boolean formula in conjunctive normal form (CNF), a conjunction of clauses where each clause is a disjunction of literals. Each literal is a variable and a sign. The literal is said to be negative if negated and positive otherwise. Universals and existentials are those variables quantified by a universal or existential quantifier respectively. A QBF of the form $\exists v_1 Q_2v_2...Q_nv_nC$ is true if either $Q_2v_2...Q_nv_nC[v_1 := T]$ or $Q_2v_2...Q_nv_nC[v_1 := F]$ is true, where T represents true and F represents false. Similarly, a QBF of the form $\forall v_1 Q_2v_2...Q_nv_nC$ is true if both $Q_2v_2...Q_nv_nC[v_1 := T]$ and $Q_2v_2...Q_nv_nC[v_1 := F]$ are true.

A QBF is trivially *false* if it contains an all-universal clause or an empty clause. An all-universal clause is a clause in which all the literals are universal. Such a clause is false since we must be able to satisfy the formula for all valuations of universals, including the combination which makes all literals false in the all universal clause. This is true so long as the clause is not a tautology, which we remove with preprocessing. A QBF is trivially *true* if it consists of an empty set of clauses.

The basic procedure for solving QBF's, [2], is similar to the DPLL procedure for SAT [3]. The order of variables is now determined by the order in which the variables occur in the prefix. Secondly, when the formula is found true, the procedure backtracks to the most recently assigned universal that has not had both true and false values explored. Further propagation is possible using unit or single existential clauses. A unit or single-existential clause is defined to be a clause in which only one existential variable is left to be assigned and all universal variables are quantified after the existential in the quantifier prefix. In this clause, the existential must be assigned true to avoid an all-universal clause.

Conflict Directed Backjumping [8] is not covered in this paper. For a detailed description of Conflict Directed Backjumping for QBF see [6]. **Solution Directed Backjumping** can be performed during QBF search when assignments result in an empty set of clauses. The first stage is to find a small set of universal literals from the current assignment which leaves the formula satisfied. This is known as a reason for the solution or the solution set. To find the solution set, each clause is examined in turn. Where a clause is satisfied by at least one existential, nothing is added to the solution set. Where a clause is only satisfied by a universal assignment, we choose one of the universals in the clause to be added to the set. We only backtrack on a universal variable when it occurs in the solution set. When both true and false assignments to a universal result in the QBF formula being true, we can combine the solution sets from both these assignments and pass the new solution set back to be used for universals assigned earlier. For more details see [6].

The idea behind **solution learning** is simple. The solution sets which are generated by solution directed backjumping are stored at certain points in the search. These then guide search away from previously visited solutions and thus reduce the amount of search performed. The solution set cannot be learned as it is. Solution sets were introduced as only containing universals. These universals come from the clauses that are not satisfied by any existentials. For solution learning, the solution set must be augmented with a subset of the existentials that satisfy the remaining clauses. This is because the solution is not caused only by the universals. Only universals are added to the solution set in solution directed backjumping because solution backtracking is only performed on universals.

In [5], solution learning was first introduced. It was realised here that not all solution sets could be learned as they were. The criterion for learning a solution set is that the solution set must be prefix closed. This means that for any variable v that occurs in the solution set, every variable quantified before v in the prefix must also occur. This will not always be the case in the solution set and so not all solution sets are learnable.

In [10], solution learning is performed using consensus for DNF cubes (the dual of resolution of CNF clauses). The QBF CNF formula C is extended to initially contain a DNF formula D such that $C \iff C \vee D$. Initially D is empty and is thus false. When a new solution is learned, it is added to D as a DNF cube. In conflict learning by resolution for SAT, resolution is performed on the clause that became empty, e, to cause the conflict and the clause that was a unit clause immediately before the last variable was assigned, u. The result of this is then resolved with the clause that became unit before u. This continues until an assignment is encountered that was not performed by propagation. At this point, the resolvent is added as a clause. Solution learning proceeds in the same way, except that the initial cube does not exist. This is instead created from a solution set. Where solution learning is referred to in the remainder of this paper, it is this method that will be considered.

We contrast '**local solution learning**' with existing QBF solution learning techniques, which we call **global solution learning**. In global solution learning [5, 10], the solution set is augmented with existentials. Otherwise, the solution set would only be valid under certain conditions. Hence a DNF cube generated from the solution set would have to be removed if these conditions change. In local solution learning, the learned cube must be removed when one of the existential variables is changed that is quantified outside one of the universal variables in the cube. This ensures that local learning only holds valid learned cubes. This is the 'local' part of the learning; the learned information is only valid in a local sub-tree in the search tree. Unless a learned cube removal system (such as relevance or size learning) is used, global learning will keep all learned cubes until the end of the search. This can result in an exponential increase in the size of the formula. Local solution learning can be considered a learned cube removal system. The size of the learned cubes will also be smaller in local solution learning, since they do not contain any existentials.

3 Complete Local Solution Learning

We call existing solution learning techniques '**incomplete**'. This indicates the fact that they do not learn all the information possible from each solution found. The methods we introduce are '**complete**', in that any solution is reused directly whenever possible, avoiding as much search as possible within the scope that solutions are preserved. The two methods differ in the scope within which found solutions are preserved, one being local and the other global. We introduce the local method first as the global method is based on it but is more complicated.[3]

QBFs of the form $\forall\exists B$. For simplicity, we first discuss the complete local solution learning for formulae which have a prefix of the form $\forall\exists$.

We use an example to show what incomplete learning misses. Consider the formula $\forall u_1 \forall u_2 \ldots \forall u_n \exists e_1 \ldots \exists e_m [(u_1 \vee u_2 \vee e_1) \wedge (u_3 \vee u_4 \vee e_2) \wedge \ldots]$. Say the formula is satisfied by the assignments where $e_1 = e_2 = F$, so that the two clauses are only satisfied by universals. Let the assignments to all the existentials in the current assignment be called E_1. Say now that the universal variables are all assigned true in the order u_1, u_2, u_3, u_4. So there are four possible solution sets that can be built. These are $\{u_1, u_3\}$, $\{u_1, u_4\}$, $\{u_2, u_3\}$ and $\{u_2, u_4\}$. Incomplete solution learning will only learn one of these sets. Yet, if we ever find ourselves assigning the universals true from *any* of the four sets, E_1 can still be used to satisfy the remaining clauses. Incomplete learning methods cannot exploit this fact, leading to unnecessary backtracks. Redundant search could be avoided in solution learning if all of the possible solution sets were learned at this stage, but this has the potential for causing an exponential increase in the size of the formula. Suppose

[3] Complete Local Solution Learning is the same algorithm as GR-Learning reported in our technical report [4], but we hope with a more descriptive name. It has been reimplemented, however, so results in this paper are new.

instead that the assignments to the universals were $u_1 = T, u_2 = T, u_3 = T, u_4 = F$. There are now two solution sets available: $\{u_1, u_3\}$ and $\{u_2, u_3\}$. Now when backjumping assigns $u_3 = F$, the algorithm will be free to choose $u_4 = T$ to satisfy the second clause. Again the original existential assignment E_1 can be used to satisfy the remaining clauses. Even if existing solution learning methods were to learn all the solution sets, this situation would not be covered. This is because solution learning is restricted to the set of universals that have been assigned.

In general, let the set of clauses not satisfied by an existential be C_u. As long as all the clauses in C_u are satisfied by some universal assignment, the remaining clauses can be satisfied by the existential assignment E_1. The only cases to be considered are those in which one or more of the clauses in C_u are not satisfied by any universal assignment. The new contribution of complete solution learning methods is to learn that the formula is satisfied so long as every clause in C_u is satisfied by a universal variable. We now describe how we can achieve this in one combined learning operation which does not increase the formula size exponentially.

Let U_i be the ith clause with the existentials removed. Let $C_u = \{c_p, c_q, \ldots\}$. We are to learn that so long as $L_n = U_p \wedge U_q \wedge \ldots$ is true, the original formula is true. L_n is not in DNF, making it hard to use standard learning methods. Note that in QBF learning, the original formula is true iff the DNF formula is true. Therefore, the QBF solver tries to make the DNF false to ensure that only new parts of the tree are explored. To put L_n in DNF, we replace each U_i by an 'indicator' variable I_i, using a simple encoding trick. We encode that a variable I_i is assigned true if and only if one of the universal variables in clause i satisfies the clause. This is encoded as several DNF cubes which can be added to the original formula before search begins. If $U_i = (u_1 \vee u_2 \vee \ldots \vee u_n)$, the cubes added will be $(\overline{u}_1 \wedge \overline{u}_2 \wedge \ldots \wedge \overline{u}_n \wedge I_i) \vee (u_1 \wedge \overline{I}_i) \vee (u_2 \wedge \overline{I}_n) \vee \ldots \vee (u_n \wedge \overline{I}_i)$. This formula shall be known as the indicator formula. So $L_n = I_p \wedge I_q \wedge \ldots$ can be added as a DNF cube when learning is performed. DNF unit propagation attempts to make the DNF formula false, and so the correct assignments are made. The indicator variables must be quantified after the variables in U_i, so that it is not possible to assign them before the variables which they indicate. The indicator variables must be universally quantified, as otherwise the cubes in the indicator formula would all be single-universal cubes. This would mean that the universals in the cubes would be assigned by propagation. The assignments of the correct value to the indicator variables is done by unit propagation in the DNF formula [10].

In solution learning, it is guaranteed that the addition of the DNF cubes does not affect the satisfiability of the formula i.e. $C \iff C \vee D$ where C is the original CNF formula and D is the additional DNF formula. If all the universal literals in U_i are assigned false, this removes all the binary cubes in the indicator formula and forces the indicator to be false by DNF unit propagation. If one of the universal literals in U_i is true, the indicator is assigned true by DNF unit propagation. This removes the remaining cubes. Thus the indicator formula can never make the formula true by itself. When a learned cube has been added, the formula can be made true by satisfying the learned cube. This will only occur if the universal variables are assigned so as to force the indicators to the values required to satisfy the learned cube. This will mean that the assignments to the universals are such that the current state has been visited previously and so no further search is required.

Solution backjumping must be slightly modified in order to support a solution set built from indicator variables. In building a solution set, where a clause is not satisfied by an existential, the indicator is added to the solution set instead of choosing one of the universals. Thus the final solution set will represent all the equivalent solution sets for the current assignment of variables. The resulting technique for backjumping is very similar to that of conflict-directed backjumping, and is shown in the context of our local complete learning algorithm in figure 4.

Unlike standard solution learning, complete solution learning does not guarantee to use the learned cube to invert the assignment to the universal. When the universal is assigned it may cause more than one indicator to be assigned through DNF unit propagation. This means that there is a possibility that more than one unassigned indicators occur in the learned cube when it is added. So the learned cube will not necessarily be a unit or single universal cube when it is added. Therefore, the algorithm must not rely on DNF unit propagation to reassign the universal after backtracking.

Take the example $\forall u_1 \forall u_2 \ldots \forall u_n \exists e_1 \ldots \exists e_m [(u_1 \vee u_2 \vee e_1) \wedge (u_3 \vee u_4 \vee e_2) \wedge (\overline{u}_4 \vee \overline{e}_2) \ldots]$. Before search begins, the formula is augmented with the indicator formula $(\overline{u}_1 \wedge \overline{u}_2 \wedge I_1) \vee (u_1 \wedge \overline{I}_1) \vee (u_2 \wedge \overline{I}_1) \vee (\overline{u}_3 \wedge \overline{u}_4 \wedge I_2) \vee (u_3 \wedge \overline{I}_2) \vee (u_4 \wedge \overline{I}_2) \vee (\overline{u}_4 \wedge \overline{I}_3) \vee (u_4 \wedge I_3)$. The indicators are quantified in a universal quantifier at the end of the formula. Now the assignments are made as follows: $u_1 = T, (I_1 = T), u_2 = T, u_3 = T, (I_2 = T), u_4 = F, (I_3 = F), E_1$, where E_1 is an assignment to the existentials such that $e_1 = F, e_2 = F \in E_1$. The assignments in brackets are those which are performed by unit propagation following the previous assignment. The solution set is built to contain $\{I_1, I_2\}$. Backjumping is performed to return to I_2. I_2 is removed and replaced with u_3. Backjumping then returns and unassigns u_3. The solution set now consists of $\{I_1\}$. Now the cube $(I_1 \wedge I_2)$ is learned, adding the indicators relevant to u_3 that were removed from the solution set during backtracking. As u_1 and u_2 are still assigned true, I_1 is true. I_2 must be assigned false by unit propagation. So u_3 and u_4 are assigned false by unit propagation. E_1 is no longer sufficient to satisfy the remaining formula so a new existential assignment must be found if the formula is to be proved true. So a new assignment E_2 is deduced such that $e_1 = F, e_2 = T \in E_2$. This results in the creation of the solution set $\{I_1, I_3\}$. Backjumping replaces I_3 with u_4 and returns to u_4. As both assignments have been made to u_4, the solution sets are combined to make $\{I_1\}$. Backjumping returns to the assignment of I_1 replacing it with u_1. The cube (I_1) is learned. This is a unit cube and thus I_1 is assigned false. This in turn assigns u_1 and u_2 false. This also removes the previously learned cube $(I_1 \wedge I_2)$. The algorithm is now free to choose values for u_3 and u_4.

General QBFs The complete learning method described above only works when the formula prefix is of the form $\forall\exists$. If there is an existential quantified outside the universal, the learned cubes can become invalid. Our first solution to this is *local*: a learned cube can remain so long as no assignment to an existential quantified outside the universals indicated in the cube is changed. If such an existential assignment is changed, the learned cube is simply discarded.

If there is more than one level of universal quantification, the situation becomes further complicated. It becomes important to distinguish between the quantification levels of the universals. Take the quantifier sequence $\forall U_1 \exists E_2 \forall U_3 \exists E_4$ where U_1, E_2, U_3 and E_4 are sets of variables. The assignments to the variables in U_1 do not depend on any existential assignments. The assignments to the variables in U_3, however, depend on the assignments to the variables in E_2. Therefore, any locally learned cube that depends on a variable in U_3 must be deleted if a variable in E_2 is changed. A locally learned cube that only depends on variables from U_1 does not need to be deleted if E_2 is changed. Therefore we distinguish between the level of quantification of the information learned, by splitting the indicator variables for each clause. There is now one indicator variable for each level of universal quantification for each clause. The indicator variable $I_{i,j}$ is the indicator variable for the ith clause and the universals quantified in the jth quantifier. The rest of the procedure continues as before. Now, only universals that are quantified at the same level in a clause are considered to be equivalent. For example, take a QBF with a prefix $\forall u_1, u_2 \exists e_3 \forall u_4, u_5$ and a clause $(u_1 \vee u_2 \vee e_3 \vee u_4 \vee u_5)$. The indicator clauses added would be $(\overline{u}_1 \wedge \overline{u}_2 \wedge I_{1,1}) \vee (u_1 \wedge \overline{I}_{1,1}) \vee (u_2 \wedge \overline{I}_{1,1}) \vee (\overline{u}_4 \wedge \overline{u}_5 \wedge I_{1,2}) \vee (u_4 \wedge \overline{I}_{1,2}) \vee (u_5 \wedge \overline{I}_{1,2})$. So two indicators exist for the one clause.

When building a solution set, the algorithm may now be faced with two or more levels of universals to choose from. It makes sense to choose the outermost level of quantification when such a choice arises, since choosing the innermost level would mean that the variables in the outermost level would still satisfy the clause.

The main points of complete local solution learning are as follows:

1. For each clause, a disjunction of the universal literals with the same level of quantification are made equivalent to an indicator variable.
2. The indicator variables are all quantified in a universal quantifier at the end of the prefix.
3. All the indicator equivalences are encoded in a DNF formula and DNF unit propagation is used to enforce the equivalences.
4. When a solution is found, one indicator is put in the solution set for each clause not satisfied by an existential. This is the indicator for the outermost universal variable that satisfies the clause.
5. When backjumping is performed, and an indicator is unassigned, the indicator is replaced by one of the universals it indicates. This indicator is remembered.
6. When backjumping concludes with the reassignment of a universal, this universal is removed from the solution set and replaced by the indicators that are relevant to the universal and that were removed during backjumping. The solution set (consisting now only of indicators) is then added as a DNF cube.
7. When backjumping merges the solution sets for both assignments to a universal variable, all the indicators of the universal variable are removed from the resultant solution set.

Figures 1 - 4 show the important details of the algorithm. Figure 1 shows the top level of the algorithm. The main differences between this and the basic QBF algorithm are as follows. Firstly, the indicator formula must be generated. Secondly, as this is a local learning algorithm, any invalid cubes must be removed after backtracking. This is automatically handled by `CompleteLocalLearn` on a true backtrack but must be done explicitly after a false backtrack.

```
CompleteLocalSolutionLearn($PB$)
P(B ∨ D) := GenerateIndicatorFormula($PB$);
do
    result := propagate($P(B \vee D)$);
    if (result = UNDEF)
        (variable, value) := nextvariable($P(B \vee D)$);
    else if (result = TRUE)
        (variable, value) := CompleteLocalLearn($P(B \vee D)$);
    else if (result = FALSE)
        (variable, value) := falselearn($P(B \vee D)$);
        $S :=$ the set of learned cubes in D that contain indicators
            that indicate variables quantified inside variable;
        $D := D - S$;
    if (variable $\neq$ -1)
        $P(B \vee D)$ := $P(B \vee D)$[variable := value];
while (variable $\neq$ -1)
return result;
```

Fig. 1. The Algorithm for Complete Local Solution Learning. P is a prefix of quantifiers, B is a Boolean formula in CNF and D is a Boolean formula in DNF.

Figure 2 shows the details of the `CompleteLocalLearn` function. This calculates an indicator set, performs backjumping, then learns a new cube at the backtracking point. The learned cubes that are no longer valid after this backtrack are removed. Figure 3 shows the function for calculating the initial indicator set for backjumping. This is similar to calculating a solution set except that a universal variable chosen for the set is replaced by its indicator variable. Figure 4 shows how the indicator set is used by the solution directed backjumping procedure. For any variable assigned by unit propagation, the variable is replaced by the variables in the cube that caused the variable to be assigned. Otherwise, if the variable has been backtracked to before, the indicator set obtained on the previous backtrack is joined with the new indicator set and the variable is removed. If the variable has not been backtracked on, it is now used as the backtrack point.

```
CompleteLocalLearn($P(B \vee D)$)
$S$ := getIndicatorSet($P(B \vee D)$);
(variable, value, $R$) := clsbackjump($P(B \vee D)$, $S$);
$R' := \{r | r$ is an indicator of variable and $r \in R\}$;
$S := S \cup R$;
$L$ := set of learned cubes in $D$ that contain indicators
    that indicate variables quantified inside variable;
$D := D - L$;
$D := D + (\bigwedge_{s_i \in S} s_i)$;
return (variable, value);
```

Fig. 2. The algorithm for learning the solutions.

4 Complete Global Solution Learning

Local solution learning requires that learned information is thrown away when it is no longer valid. The use of only the universals in the solution sets means that the

```
getIndicatorSet($P(B \vee D)$)
if $D$ is true
    return the set of indicators in the cube that satisfies $D$;
else
    $S := \{\}$;
    for each clause $c$ in $B$
        if $c$ is not satisfied by an existential
            $v :=$ the outermost universal that satisfies the clause;
            $l :=$ the level of quantification of $v$;
            $I :=$ the indicator for clause $c$, level $l$;
            $S := S \cup \{I\}$;
    return $S$;
```

Fig. 3. The algorithm for calculating the indicator sets.

```
clsbackjump($P(B \vee D)$, $S$)
$R := \{\}$;
while there is backtracking to be done
    variable := the last assigned variable;
    $P(B \vee D) := P(B \vee D)$[variable unassigned];
    if variable $\in S$ and variable $\in \forall$
        if $l$ was assigned by single universal propagation
            $c :=$ the cube that caused the assignment of $l$;
            Add to every literal in $c$ to $S$;
            Remove variable from $S$;
            if variable is an indicator variable
                Add variable to $R$;
        else if both values of variable have been assigned
            $S' :=$ the indicator set from the last assignment of variable;
            $S := S \cup S'$;
        else
            value := value assigned to variable;
            return ($l$, value, $R$);
return (-1, false, $\{\}$);
```

Fig. 4. Algorithm for backjumping using indicator sets. S is an indicator solution set.

learned information can become invalid when some existentials are changed. The simplest way to overcome this problem is to add some existentials to the solution set. This is done in the same way as is done with solution learning. This way, the solutions can be kept beyond the reassignment of an existential. This means that the same solution will be avoided if it occurs later in the search. Take, for example, the formula $\exists e_1 \forall u_2, u_3 \exists e_4 \forall u_5, u_6 \exists e_7[(e_4 \vee u_5 \vee u_6 \vee e_7) \wedge (\overline{e}_4 \vee \overline{u}_5) \wedge \ldots]$ and that there is an indicator $I_{1,4}$ for the first clause shown such that $I_{1,4} \iff u_5 \vee u_6$. Say that e_1, u_2, u_3, u_5 and u_6 are assigned true and that e_4 and e_7 are assigned false and that the remaining clauses are satisfied by the existentials. So now the solution set is built and $I_{1,4}$ is added to the set. Additionally, e_4 is added to the solution set as it satisfies the second clause. If e_1 does not satisfy any clause on its own then it does not need to be added to the solution set. So the cube learned is $(I_{1,4} \wedge \overline{e}_4)$. Now if later, e_4 is assigned true, this cube will be removed and so it cannot cause the DNF formula to become true. If later still, e_4 is unassigned, this cube will become active again. If e_4 is again assigned false, DNF unit propagation will assign $I_{1,4}$ false, resulting in u_5 and u_6 being assigned false. Hence, the same solution will not be explored again.

There is a second choice to make when considering the existentials that satisfy the clause. Only one existential is needed from each clause that is satisfied by an existential. In both normal solution learning and complete global solution learning, it makes sense to choose the innermost existential that satisfies a clause for the solution set when presented with a choice. Universals in a clause that are quantified inside all existentials in that clause can be removed [1]. Similarly, existentials in a cube that are quantified inside all universals in that cube can be removed [10]. By choosing the innermost existential that satisfies a clause at the same time as choosing the outermost universal where no existential satisfies a clause, there is more chance that the existential will be quantified inside the universals in the solution set. Therefore, the innermost existentials can be removed and so the added cube will be smaller and easier to manage.

There are reasons why local learning might be better than global learning. First, the learned information is dynamically deleted in local learning. Therefore, the size of the formula is less likely to grow exponentially in size. In global learning, learned information is kept for the duration of the search. However, two methods have been described [5] which dynamically delete the learned information. These are size bounded and relevance bounded learning. Local learning does not need these methods as it contains its own dynamic deletion system. If size and relevance bounding prove to be the best methods for controlling size growth of the formula then global learning will prove to be more useful. It may prove, however, that solutions do not reoccur outside the local branch being explored. Solutions depend on more information than conflicts in general; a conflict only requires one clause to become empty whereas a solution requires all clauses to be removed. Therefore, it is less likely that solutions will reoccur in different parts of the search tree, and local learning might prove to be the better learning method.

For space reasons we omit pseudocode for the global learning functions, as they are very similar to those for local learning. For global learning the algorithm is almost identical to that in figure 1. The main change is to call the global learning function CompleteGlobalLearn instead of CompleteLocalLearn on line 7. Also, the learned cubes are not discarded after an existential is reassigned, i.e. lines 10 & 11 are omitted. The global learning function CompleteGlobalLearn is also almost identical to CompleteLocalLearn shown in figure 2. As with the main function, the learned cubes are not removed: in this case we omit lines 5 & 6 of the previous function. Finally, the indicator set is built in the same way as for local learning, except now, if a clause is satisfied by an existential, the innermost existential that satisfies the clause is added to the solution set. That is, we omit lines 11, 12 & 13.

5 Experimental Evaluation

In the experiments presented here, the hypothesis will be clearly stated so that the experimental analysis is given more direction. On occasion, our original hypothesis will be wrong and corrected in discussion.

In all the implementations of algorithms presented, the same QBF solving library was used. The solving library performs all the basic tasks of the QBF solver. This includes reading the QBF instance from a file and initialising the data structures, assigning and unassigning variables and performing propagation steps. This avoids any differences between algorithms other than the intended differences. The library and all the solvers were written in C++ and compiled using GNU gcc 3.2. The compiler was passed the -O3 flag for optimisation of the code and the -static flag to statically link the libraries used. When a choice of variables is available the solvers all use the same heuristic, choosing the unassigned variable from the outermost quantifier which has the smallest variable index. The solvers were run on a cluster of Intel Pentium II 450 Mhz computers with 386 MB RAM running Redhat Linux 7.1. Each algorithm was given a timeout of 1200 seconds on each problem. The instances used were the benchmark instances available from QBFLib (www.qbflib.org) as of 15th March 2004.

Two types of graphs are presented, comparing the the number of backtracks of two algorithms, and comparing the run time. In a backtrack comparison, the number of backtracks performed by the base algorithm is plotted on the x-axis. The y-axis shows the improvement factor gained from using the second algorithm. This is calculated by dividing the number of backtracks performed by the algorithm without the new idea by the number performed by the algorithm with the new idea. A line is drawn at a value of 1 on the y-axis. This is called the equality line. Points which lie on this line are where both algorithms performed the same number of backtracks. A point which lies above the equality line is where the second algorithm performs less backtracks than the base algorithm on an instance. The distance of the point from the equality line indicates how much better or worse the algorithm with the new idea performed compared to the algorithm without the new idea. No points are plotted where either algorithm did not complete before the time-out. In a run time comparison, the graph is similar. This graph shows points where one algorithm timed out on an instance. Additionally, a diagonal line is drawn on the graph starting below the equality line and meeting the equality line at a value of 1200 seconds. This is known as the time-out line. A point which lies on this line is where the comparison algorithm timed-out but the base algorithm did not. Points which lie at 1200 seconds on the x-axis above the equality lines are ones where the base algorithm timed out but the comparison algorithm did not.

In order to evaluate the performance of the new solution learning techniques, each is compared to an algorithm that uses solution directed backjumping. Four QBF algorithms were implemented. These were CLearn, CSLearn, CLSLearn and CGSLearn. CLearn implements conflict learning and solution backjumping. CSLearn implements conflict and incomplete solution learning. CLSLearn implements conflict learning and complete local solution learning. CGSLearn implements conflict learning and complete global solution learning. We used CLearn and CSLearn as reference implementations, as they have all the same data structures, heuristics, etc, so run time and backtrack counts can be compared directly. In all comparisons, the number of solution backtracks is used. This is because the solution learning algorithms should directly affect the number of solution backtracks. The run time is also used in the comparisons. As new cubes are added to the QBF, more work must be done in assigning each variable. This work will directly affect the run time. It may not be worth doing the additional work in reducing the number of backtracks if more time is taken overall. In the complete learning techniques, some cubes are added during preprocessing. There may be a better way of setting up the indicators. An example of this is the use of an implicit assignment of the indicator variables. Therefore, the run time measure of these algorithms, whilst presented, does not necessarily reflect the best obtainable performance of the algorithm.

Our first experiments compared the existing and our new solution learning techniques with solution backjumping. All algorithms implement conflict learning.

Hypothesis

1. Solution learning performs the same or less solution backtracks than solution directed backjumping on a QBF benchmark instance.

2. Complete local solution learning performs the same or less solution backtracks than solution directed backjumping on a QBF benchmark instance.
3. Complete global solution learning performs the same or less solution backtracks than solution directed backjumping on a QBF benchmark instance.

Unfortunately, space precludes us from presenting our results in detail, but we summarise them as they inform the next set of experiments. Standard solution learning can perform less universal backtracks than solution backjumping. Unfortunately, there are very few points for which the difference between the algorithms is significant. As solution learning is just an overhead in most cases, using solution learning can result in the algorithm timing out where it would have completed if solution learning was not used. It is very rare when solution learning does complete faster than solution backjumping. Complete local solution learning performs better that solution backjumping on many problems. There are still many points for which complete local solution learning (CLSLearn) is an overhead overall. CLSLearn can sometimes solve problems that solution backjumping cannot within the timeout. Complete global solution learning performs better than solution backjumping on many problems. However, there are no points where complete global solution learning can solve problems that solution backjumping cannot but there are points that solution backjumping can solve that complete global solution learning cannot. It is unlikely that complete global solution learning is much better than incomplete solution learning when the run time is compared.

We are now more positive about complete local solution learning than in our previous report [4]. We ascribe this to more efficient implementation. This also suggests that performance of solution backjumping and learning techniques is critically dependent on small features of implementation, such as choice of solution sets in backjumping.

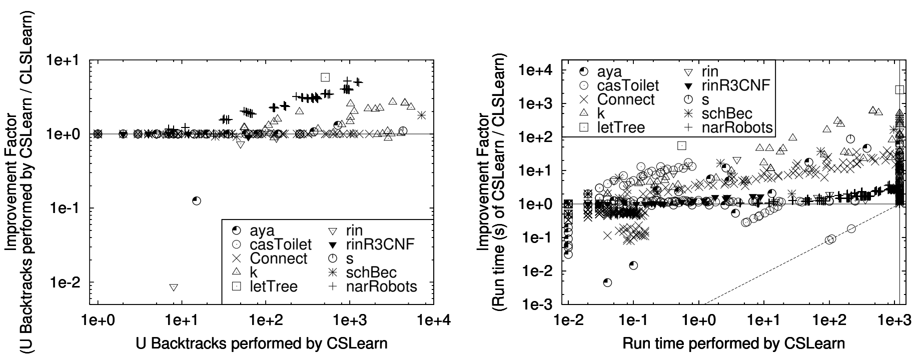

Fig. 5. A comparison of the number of universal backtracks (left) and run time (right) performed by solution learning and CLSLearn. For a description of the presentation of the graphs, see the main text.

We next compare our new complete solution learning techniques, i.e. CLSLearn and CGSLearn, with the incomplete solution learning technique, i.e. CSLearn.

Hypothesis

1. Complete local solution learning performs the same or less universal backtracks than solution learning on a QBF benchmark instance.
2. Complete global solution learning performs the same or less universal backtracks than solution learning on a QBF benchmark instance.

Figure 5(left) shows that the improvement factor is often better than 1, though never reaching 10. Most points lie on or close to the equality line, with just 2 showing a significant deterioration. Figure 5(right) shows many points with an improvement factor better than one, and many that this enables CLSLearn to solve that CSLearn cannot. The best shows an improvement factor of over three orders of magnitude. There are three points that lie on the time-out line of CLSLearn.

Figure 6(left) is very similar to figure 5(left). Figure 6(right) shows a much smaller improvement in run time and many more cases where run time increases through the use of CGSLearn, compared to CLSLearn.

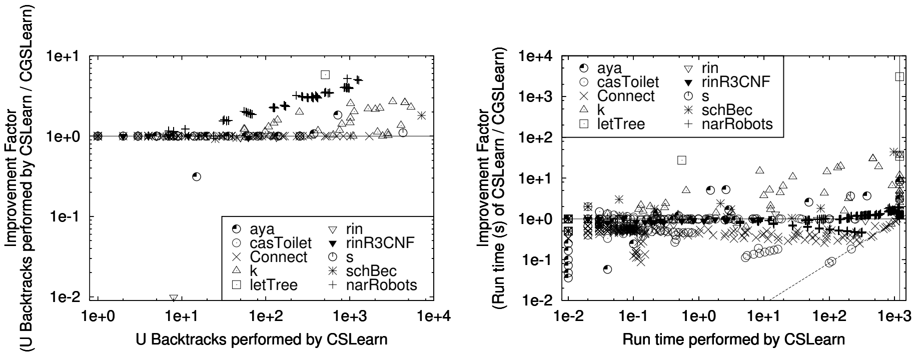

Fig. 6. A comparison of the number of universal backtracks (left) and run time (right) performed by solution learning and complete global solution learning.

The results show that CLSLearn almost always performs the same or less universal backtracks than the incomplete solution learning technique. However, there are some points where this is not the case. This means that the first part of the hypothesis must be rejected. These points are likely to be caused by the CLSLearn technique missing out on finding a particularly good solution set and therefore performing more solution backtracks. As both the incomplete solution learning technique and CLSLearn both cause some overheads, the run time graph is much different to the comparison with solution backjumping. CLSLearn does cause some additional overheads compared to the incomplete solution learning technique. Therefore there are some instances that are not solved by CLSLearn but are solved by incomplete solution learning. There are many more points that can now be solved by CLSLearn that incomplete solution learning could not solve before the time-out. This shows the importance of learning more of the solutions in a solution learning technique.

The results for complete global solution learning (CGSLearn) are similar. Again, the second part of the hypothesis must be rejected. CGSLearn has even more overheads, as reflected in the run time comparison. There are more points where solution learning solves the instances but CGSLearn does not, and fewer points where CGSLearn solves the instance but incomplete solution learning does not. These points show that the complete learning technique is still worth doing. What is less obvious is whether it is worth performing global learning compared to performing local learning. This is explored in the next experiment.

Hypothesis

1. Complete global solution learning performs the same or less universal backtracks than complete local solution learning on a QBF benchmark instance.

Figure 7(left) shows some points with an improvement, but none with an improvement factor greater than 2.5. Figure 7(right) shows that the overheads cause most instances to deteriorate in run time, suggesting that CLSLearn is more effective than CGSLearn.

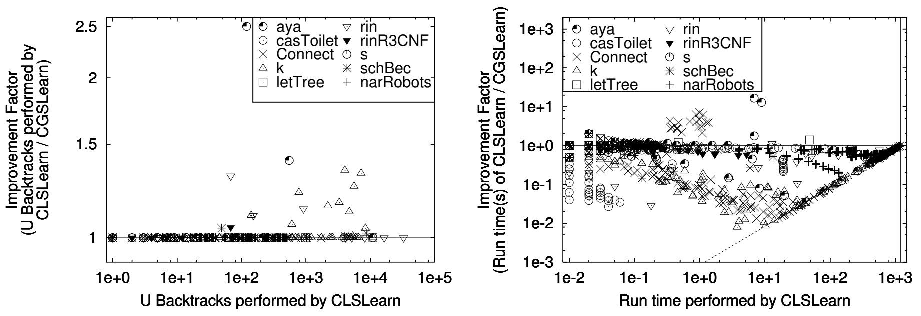

Fig. 7. A comparison of the number of universal backtracks (left) and run time (right) performed by CLSLearn and CGSLearn.

The results show that the hypothesis is correct as CGSLearn never performs more universal backtracks than CLSLearn. However, only small improvements are obtained. As CGSLearn keeps the solutions for the remainder of the search, more overheads are caused than with CLSLearn. This is reflected in the run time where CGSLearn cannot solve many instances that CLSLearn can. There are no instances where CGSLearn solves the instance but CLSLearn does not. Therefore, it appears that it is not worth performing the global learning compared to local learning.

6 Conclusions

We presented two new techniques for solution learning. The first learns more information than would be learned by the incomplete solution learning technique. This is done without an exponential increase in formula size per solution despite the learning of an exponential number of solutions. This is achieved by using a simple but effective encoding trick. The second technique extends the first technique by allowing the learned solutions to be stored for the duration of the search.

Our experimental analysis shows that complete local solution learning performs better than the existing solution learning technique in most cases. Our second learning technique, complete global solution learning, does not perform as well, but still does better than the incomplete solution learning technique on many instances.

The implementations of the complete solution learning algorithms presented are not likely to be optimal. The encoding trick used in the solution learning adds to the original formula. This could be avoided by holding this information implicitly in the data structures of the QBF, or by only adding the indicator formula when it is needed. This may result in better operation of the complete solution learning techniques. It is worth first testing the hypothesis that the time taken to solve a QBF benchmark problem with indicator cubes is more than that taken to solve the QBF benchmark problem alone. This is likely to be the case since the indicator formula must be handled in addition to the original formula. This would also show how much more work must be done by the solver to deal with the indicator formula. It would also be interesting to try size and relevance bounding with the complete global solution learning technique.

Acknowledgements

The first author is supported by a Royal Society of Edinburgh SEELLD Support Fellowship and the second was supported by EPSRC grant GR/55382. We thank other members of the APES and CPPOD research groups, and Massimo Narizzano. We are very grateful to referees of SAT 05 for their helpful comments.

References

1. H. Kleine Buning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *information and Computation*, 117:12–18, 1995.
2. M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
3. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
4. I. Gent and A. Rowley. Solution learning and solution directed backjumping revisited. Technical Report APES-80-2004, APES Research Group, 2004. http://www.dcs.st-and.ac.uk/~apes/apesreports.html.
5. E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for quantified Boolean logic satisfiability. In *Proc. AAAI 2002*, pages 649–654. AAAI Press, 2002.
6. E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified Boolean logic satisfiability. *Artificial Intelligence*, 145(1–2):99–120, 2003.
7. R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 160–175. Springer, 2002.
8. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.
9. L. Zhang and S. Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 442–449. ACM Press, 2002.
10. L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In *Proc. CP 2002*, pages 200–215. Springer, 2002.