Solving 2-CNF Quantified Boolean Formulae using Variable Assignment and Propagation

Ian P. Gent and Andrew G. D. Rowley

University of St. Andrews
Dept. of Computer Science
{ipg, agdr}@dcs.st-and.ac.uk

Abstract. In this paper we introduce a new algorithm for Quantified Boolean Formulae in 2-CNF. This class is already known to be solvable in linear time, but our algorithm is new in being based on unit propagation. The algorithm has quadratic time complexity. We prove the correctness of this algorithm and report experimental results on its performance in practice. We then go on to show a possible method of integration of this algorithm for propagation in a general QBF solver, and experimentally verify and compare this new algorithm to a current algorithm for general QBFs. Experimentally, our new algorithm shows significant run time improvements. While these results are provisional, it shows that the integration of more powerful techniques into QBF algorithms deserves further research.

1 Introduction

There has been recent interest in providing algorithms for 2-SAT based on unit propagation [1]. However, although the problem is in the same complexity class [2], there has not been so much interest in the satisfiability of quantified Boolean formulae with two literals per clause (2-CNF QBFs). We correct this by showing how a unit-propagation based algorithm can solve 2-CNF QBF's albeit not in linear time. Since the existing algorithm for 2-CNF QBFs is based on a graph algorithm, our algorithm based on unit propagation should be more easily integratable into general QBF algorithms.

The purpose of this paper is to begin to mirror the work done for 2-SAT solving with 2-CNF QBF solvers. The paper is structured as follows. Firstly, we give an introduction to 2-CNF QBFs, focussing on conditions for solvability presented by Aspvall et al [2]. We then present the Backtrack-Once algorithm as a base for solvers and prove its correctness. We next show an algorithm that integrates this with a general backtracking QBF solver. Finally, we perform an experimental analysis on implementations of the algorithms.

2 Background

A 2-CNF QBF is a quantified Boolean formula of the form

$$Q[q_1, q_2, \ldots]F[x_1, x_2, x_3, \ldots].$$

Q is a sequence of quantifier sets $q_1, ...q_n$, each of which is either existential or universal. Each of the quantifier sets quantifies the variables and each variable is in exactly one set. F is a Boolean formula in conjunctive normal form, where each clause contains exactly two literals. This means that, initially at least, unit clauses and empty clauses cannot exist in the QBF.

In [2], a 2-CNF QBF is converted into an equivalent graph, with a vertex for each literal v and its complement $\bar{v}$. Each clause (x, y) is then converted into a pair of edges $\bar{x} \rightarrow y$ and $\bar{y} \rightarrow x$. This reflects the fact that if x is assigned false, y must be assigned true by the principle of unit propagation. When the graph is created in this way, it will have the property that if there is a path from some literal x to some other literal y, then there must be a path from $\bar{y}$ to $\bar{x}$ [2].

A strongly connected component of a graph is a maximal set of vertices such that there is a path from every vertex to every other vertex in the component. For example, a cycle in the graph is a strongly connected component.

A QBF is false iff any one of the following conditions hold

1. An existential literal e is in the same strongly connected component as its complement $\bar{e}$. This would mean that there is a path from e to $\bar{e}$ and back to e so that assigning e true forces e false and vice versa - clearly a contradiction.
2. A universal literal u, quantified by q_u, is in the same strongly connected component as an existential literal e, quantified by q_e where $e < u$. This forces e and u to have the same value. This is a contradiction since for at least one value of e, u must be allowed to be assigned both true and false.
3. There is a path from a universal literal u to another universal literal v (including $v = \bar{u}$). This forces v to have the same value of u but v must be allowed to be both true and false for any assignment of u.
4. There is a path from a universal literal to an existential literal and its complement. This implies condition 3 and follows from the definition of the graph. If there is a path $u \rightarrow e \rightarrow \bar{e}$ then there is a path $e \rightarrow \bar{e} \rightarrow \bar{u}$ and so a path $u \rightarrow e \rightarrow \bar{e} \rightarrow \bar{u}$.

Aspvall et al [2] introduced conditions 1 to 3, and proved that if none of them hold, then the QBF is satisfiable. In this paper we introduce the fourth condition, which implies condition 3. While redundant, it is useful in our proofs of correctness of the algorithm we introduce.

3 The Backtrack Once Algorithm for 2-CNF QBFs

The algorithm presented in [2] is based on searching the graph of the 2-CNF QBF to see if any of conditions 1, 2 and 3 hold. If they do not, the QBF is true. The problem with this algorithm is that it does not assign any values to the variables of the QBF. This makes it difficult to incorporate into an algorithm for generalised QBFs, as was pointed out in [1].

Figure 1 shows a simple algorithm for solving 2-CNF QBFs. This is based on the standard algorithm for 2-SAT as explained and cited in [1]. The main

```
BtoQsat(Q)
   // Q is a 2-CNF QBF
If Q = ∅
      Return true
If {} ∈ Q or {∀} ∈ Q
      Return false
v := outermost unassigned variable in Q
P := QsatPropUnit(Q[v := true])
If (v ∈ ∃ and ({} ∈ P or {∀} ∈ P)) or
      (v ∈ ∀ and ({} ∉ P and {∀} ∉ P))
      Q := QsatPropUnit(Q[v := false])
Else
      Q := P
Return BtoQsat(Q)
```

Fig. 1. Backtrack Once Algorithm. The notation $\{\forall\}$ indicates any clause containing only universal literals.

difference is the inclusion of QBF specific expressions, such as the all universal clause ($\{\forall\}$).

The algorithm chooses a variable from the outermost quantifier of the QBF. This is important here especially since we now need to check for condition 2. If we did not choose variables in this way, this algorithm would fail if any universal was in the same strongly connected component as any existential. The algorithm assigns the chosen variable and performs unit propagation. This function is provided by QsatPropUnit. This is the standard unit propagation algorithm for satisfiability, with the addition that unit propagation stops on an all universal clause. This can be done in linear time, complexity $O(m)$ for a problem with m clauses.

If, after unit propagation, a failure is detected, the assignment to an existential is complemented. If a success is detected, the assignment to a universal is complemented. In this way, the algorithm backtracks only on the current assignment, never to previous assignments. This gives it a complexity $O(nm)$ for a problem with n variables, since it may be necessary to complement the assignment to every variable.

We must now show that the algorithm is correct. We do this by using the fact that a QBF is satisfiable iff none of the conditions 1 to 4 are met.

Theorem 1. *If any one of conditions 1 to 4 are met,* BtoQsat *returns **false**.*

Proof. If condition 1 is met then there is a path from an existential e to its complement $\bar{e}$ and vice versa. Assume that the strongly connected component only contains existentials. So there must be a set of clauses

$$(\bar{e} \vee e_1) \wedge \ldots \wedge (\bar{e}_p \vee \bar{e}) \wedge (e \vee e_q) \wedge \ldots \wedge (\bar{e}_n \vee e).$$

BtoQsat will eventually assign some variable in the strongly connected component, say e. e will be assigned true and unit propagation will occur. This will force e_1 to e_p to be assigned true and so leave the clause $(\bar{e}_p \vee \bar{e})$ empty. This means that e will be assigned false, forcing e_q to e_n to be assigned false, leaving the clause $(\bar{e}_n \vee e)$ empty. This is detected on the next recursive call, as the QBF now contains an empty clause. If the strong component also contains a universal, then condition 4 is met (see below).

If condition 2 is met then there is a path from universal u to an existential e and back again to u, where e is quantified before u in Q. Since the algorithm is assigning variables from the outermost quantifier first, e will be assigned before u. Assume that the strongly connected component does not contain any universals other than u. Then there is a set of clauses

$$(\bar{e} \vee e_1) \wedge \ldots \wedge (\bar{e}_p \vee u) \wedge (\bar{u} \vee e_q) \wedge \ldots \wedge (\bar{e}_n \vee e).$$

BtoQsat will assign e true and unit propagate. This will force e_1 to e_p to be assigned true and so leave the clause $(\bar{e}_p \vee u)$ all universal. This means that e will be assigned false, forcing e_n to e_q to be assigned false, leaving the clause $(\bar{u} \vee e_q)$ all universal, which will be detected in the next recursive call. Note that if u were outside e in Q, u would be assigned first which would not necessarily lead to a contradiction, for either true or false.

If condition 3 is met then there is a path between two universals, u and v. Then there is a set of clauses

$$(\bar{u} \vee e_1) \wedge \ldots \wedge (\bar{e}_n \vee v).$$

If u is assigned first, then it will be assigned true, forcing e_1 to e_n true, leaving the clause $(\bar{e}_n \vee v)$ all universal. This will be detected in the next recursive call. If v is assigned first, e_n to e_1 will be forced false, leaving the clause $(\bar{u} \vee e_1)$ all universal. If some existential $e_p, 1 < p < n$ is assigned first, this will force e_{p+1} to e_n true, leaving an all universal clause. e_p will then be assigned false, leaving the first clause all universal.

If condition 4 is met, then condition 3 holds as we argued earlier, and the above case applies.

We have shown that the algorithm works correctly if the QBF is unsatisfiable. What we must now show is that it also works when the QBF is satisfiable.

Theorem 2. *If* BtoQsat *returns* ***false****, one of the conditions 1 to 4 is met.*

Proof. It is assumed that the 2-CNF QBF does not contain any unit or empty clauses. QsatPropUnit will stop on an empty or all universal (unit) clause if they exist after a variable assignment, but this will only occur if one of conditions 1 to 4 is met.

If the original problem has an all universal binary clause, BtoQsat will return false. Such a clause satisfies condition 3, so this is the correct result.

Otherwise, we assume that **false** is returned for a 2-CNF QBF containing n or less variables for which one of conditions 1 to 4 is met. Then, given a 2-CNF QBF containing $n+1$ variables, one of two conditions can hold.

If the outermost variable e is existential, and **false** is returned, then an empty or all universal clause must be in the problem. If e is assigned true, and unit propagation leads to an empty clause, then e is assigned false, and unit propagation again leads to an empty clause, then condition 1 must hold. If assigning e true leads to an empty clause and assigning e false leads to an all universal clause, then condition 4 must hold. If assigning e true leads to an all universal clause and then assigning e false leads to an empty clause, then again condition 4 must hold. If assigning e true and then false leads to an all universal clause for both assignments, then condition 2 or 3 must hold. If assigning e true or false does not lead to an empty or all universal clause, then the problem will be left with n or less variables and so, by induction, will fail correctly.

If the outermost variable u is universal, and **false** is returned, then an empty or all universal clause must be in the problem. If assigning u true leads to an empty clause, then condition 4 must hold. If assigning u true leads to an all universal clause, then condition 3 must hold. If assigning u true succeeds then u will be assigned false. If assigning u false leads to an empty or all universal clause, then condition 3 or 4 must hold respectively. If neither assignment to the universal results in an empty or all universal clause, then the problem will again be left with n or less variables and so will fail correctly.

4 Integrating BtoQsat for general QBF solving

Since BtoQsat is based on unit propagation, it should be relatively easy to integrate into a general QBF solver. For this purpose, the general backtracking algorithm, BTQsat [3], is a good place to begin. It does not contain any of the complications introduced by using a backjumping algorithm. The integration presented here is purely to show that it is possible, and the way in which one might go about it.

Figure 2 shows an integration of BTQsat and BtoQsat. This algorithm is not proved to be correct here, but the output of an implementation of the algorithm has been verified to be correct by comparison (see later). The proof is left for another paper.

BtoQsat has to be modified slightly for the integration, to take into account the fact that the binary clauses are not the only clauses in the problem. We must be careful not to violate the prefix ordering of the QBF, and so the binary clauses can only be used when a variable from the outermost quantifier is contained within one of the binary clauses. We must also look at all the clauses when deciding success or failure of an assignment to a variable in a binary clause. This is seen particularly in the assignment of a universal variable within a binary clause. Success is now only declared when all the clauses of the problem are satisfied by the assignment. Further details of the integration are left as the subject of another paper.

```
BTBtoQsat(Q)
If Q = ∅
    Return(true)
If {} ∈ Q or {∀} ∈ Q
    Return(false)
If Pure Existential literal e ∈ Q
    Return BTBtoQsat(Q[e := true])
If Pure Universal literal u ∈ Q
    Return BTBtoQsat(Q[u := false])
If Unit or Single Existential clause c ∈ Q
    Return BTBtoQsat(Q[c := true])
v = outermost variable in Q
If Binary clause c ∈ Q containing v
    Return BTBtoQsat-Bto(Q)
Else
    r1 := BTBtoQsat(Q[v := true])
    If (v ∈ ∃ and r1 = false) or
        (v ∈ ∀ and r1 = true)
        Return BTBtoQsat(Q[v := false])
    Else
        Return(r1)

BTBtoQsat-Bto(Q)
v = outermost variable in Q
C = set of binary clauses in Q
If Q= ∅
    Return(true)
If {} ∈ Q or {∀} ∈ Q)
    Return(false)
If C = ∅ or v ∉ C
    Return(BTBtoQsat(Q))
P := QsatPropUnit(Q[v := true])
If (v ∈ ∃ and ({} ∈ P or {∀} ∈ P)) or
    (v ∈ ∀ and P= ∅)
    Q:= QsatPropUnit(Q[v := false])
    Return(BTBtoQsat-Bto(Q))
ElseIf (v ∈ ∃ and Q= ∅) or
    (v ∈ ∀ and ({} ∈ Q or {∀} ∈ Q))
    Return(BTBtoQsat-Bto(P))
Else
    r1 := BTBtoQsat-Bto(P)
    If (v ∈ ∃ and r1 = false) or
        (v ∈ ∀ and r1 = true)
        Return BTBtoQsat-Bto(Q[v := false])
    Else
        Return(r1)
```


Fig. 2. An algorithm that uses BtoQsat for propagation. A single existential clause is a clause containing an existential e quantified by q_e and universals $u_1 \ldots u_n$ quantified by $q_1 \ldots q_n$ such that $\forall q_u e < u$.

5 Experimental Analysis

BtoQsat was implemented in C++, along with the standard backtracking algorithm for general quantified satisfiability, BTQsat, an algorithm that is similar to that described in [3]. These two algorithms were then tested on randomly generated EAE 2-CNF QBFs with between 10 and 50 variables per quantifier, in steps of 10, and with between 10 and 100 clauses, again in steps of 10, with 100 problems generated for each data point. The outputs from these algorithms was verified to be correct by comparison to the output from Rintanen's algorithm [4]. These experiments were run on an Intel Celeron 1000Mhz processor with 256MB RAM.

The algorithms were compared by the total number of variable assignments that each made on the 100 problems in each set. A variable assignment is counted when ever any variable is assigned, including unit and single existential clauses and pure literals.

Figure 3 shows this comparison with the number of variables per quantifier held at 10 (30 variables in total) and the number of clauses varied between 10 and 100. BTQsat shows the classic phase transitional behaviour, whereas BtoQsat tends towards a constant as the number of clauses increases.

Figure 4 shows the comparison with the number of clauses held at 50, and the number of variables per quantifier varied from 10 to 100 (30 to 300 variables in total). BTQsat shows a logarithmic increase on this test set, but BtoQsat now is seen to increase in a fairly linear fashion.

BTBtoQsat was also implemented in C++, and compared with BTQsat. This comparison was performed on randomly generated EAE 5-CNF QBFs with 10 variables per quantifier and between 25 and 250 clauses in steps of 25, with 100 problems for each data point. The outputs were again compared with the output from Rintanen's algorithm [4] for verification. The comparison was made with the run times of these algorithms, as opposed to the number of variables assigned.

Figure 5 shows this comparison. We see that both algorithms show the classic phase transitional behaviour as would be expected. BTBtoQsat is seen to run in less time on these random problems, showing the effect of using 2-CNF QBF algorithms for propagation.

6 Conclusions and Further Work

In this paper, we presented and verified an algorithm for solving 2-CNF QBFs based on unit propagation techniques. This algorithm, BtoQsat has complexity $O(mn)$ for a QBF with m clauses and n variables, making it worse than the graph based algorithm.

We then went on to show how this algorithm can be used for propagation in a general backtracking QBF solver. Experimentally, the integrated algorithm performed significantly better than the algorithm without 2-CNF propagation.

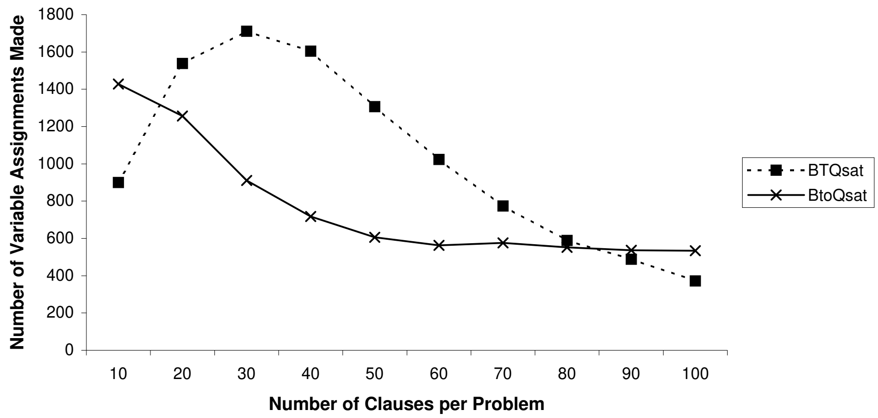

Fig. 3. A comparison of BTQSAT and BTOQSAT on random problems with 10 variables per quantifier and a varying number of clauses.

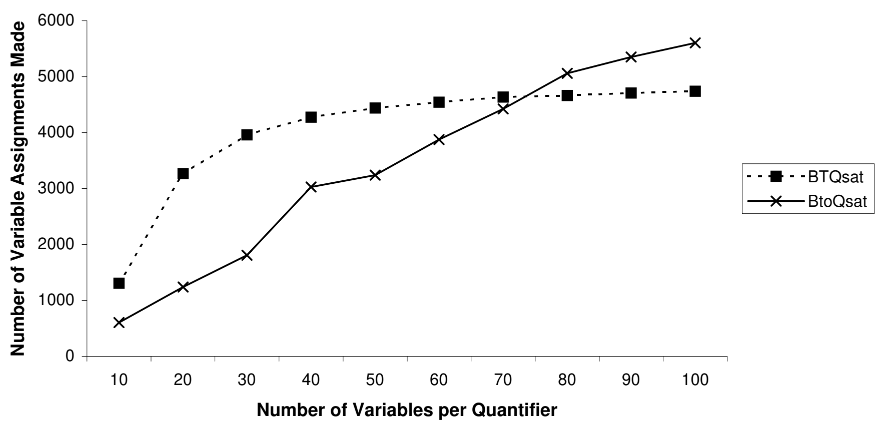

Fig. 4. A comparison of BTQSAT and BTOQSAT on random problems with 50 clauses per problem and a varying number of variables per quantifier.

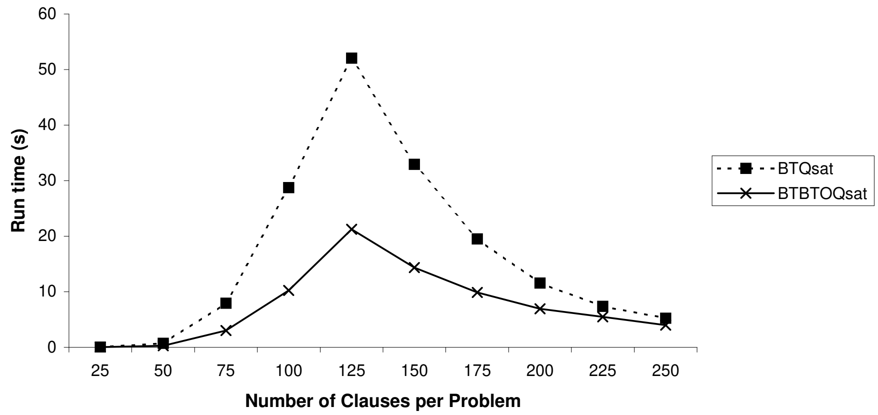

Fig. 5. A comparison of BTQSAT and BTBTOQSAT on random problems with 10 variables per quantifier and a varying number of clauses.

This warrants further research in the use of a backjumping algorithm, such as [5], in this integration.

The variables in strongly connected components of the graph of a 2-SAT problem have been found to be equivalent. It may be possible to use this fact for solving 2-CNF QBFs, by removing strongly connected components. It may also be of interest to see the role of pure, or monotone, literals in 2-CNF QBFs. These details may allow us to create a linear time algorithm for 2-CNF QBFs based on unit propagation.

Acknowledgements

This work was supported by EPSRC Grant GR/R55382. The authors are members of the APES research group `http://www.dcs.st-and.ac.uk/~apes`. We thank the other members of the research group, especially Lyndon Drake.

References

1. Val, A.: On 2-sat and renamable horn. In: AAAI-00. (2000) 279–284
2. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified Boolean formulas. Information Processing Letters **8** (1979) 121–123
3. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified Boolean formulae. In: AAAI-98. (1998) 262–267
4. Rintanen, J.: Improvements to the evaluation of quantified boolean formulae. In: IJCAI-99. (1999) 1192–1197
5. Guinchiglia, E., Narizzano, M., Tacchella, A.: Backjumping for quantified Boolean logic satisfiability. In: IJCAI-01. (2001)