

Local Search for Quantified Boolean Formulas

N. Hristov
University of West Georgia
Carrollton, Georgia, USA
nedko@cs.westga.edu

A. Remshagen
University of West Georgia
Carrollton, Georgia, USA
anja@westga.edu

ABSTRACT

Emerging applications demand to solve problems that are considered to be more difficult than NP-complete problems. Evaluating a quantified Boolean formula (QBF) is one of those problems. For the satisfiability problem (SAT), the NP-complete subclass of QBF, excellent solvers have been developed. However, research on subclasses of QBF at higher levels has only recently surged. We explore the possibility and potential of local search algorithms, which have been proved to be very successful in the case of SAT. Though local search cannot be applied directly to QBF, we introduce an approach to solve QBF at the second level of the polynomial hierarchy that profits significantly from local search.

Categories and Subject Descriptors

D.1.6 [Programming Techniques]: Logic Programming;
F.4.1 [Mathematical Logic and Formal Languages]:
Mathematical Logic—*computational logic*; F.1.3 [Computation by Abstract Devices]: Complexity Measures and Classes—*complexity hierarchies, relations among complexity classes*

General Terms

Algorithms, Theory

Keywords

Quantified Boolean formula, logic, local search, polynomial hierarchy

1. INTRODUCTION

The problem *Quantified Boolean Formula (QBF)* demands to evaluate a quantified propositional formula. Emerging applications, like, for example, conditional planning studied by Rintanen [13] or symbolic model checking explored by Plaisted, Biere, and Zhu [11], have pushed forward the work on QBF. For further applications, see for example Giunchiglia, Narizzano, Tacchella [5]. QBF is a generalization of the *satisfiability problem (SAT)* which requires to

determine whether a propositional formula is satisfiable. Although SAT is NP-complete and thus very hard, there has been tremendous success in solving SAT. SAT solvers can be divided into *complete* methods, that always terminate with the correct output, and into *incomplete* methods, that may not determine a solution.

The most successful complete methods implement backtracking search with additional improvements (Bayardo and Schrag [1], Marques-Silva and Sakallah [8], Zhang [17], Zhang, Madigan, Moskewicz, and Malik [18]). A second class of very effective SAT solvers uses local search. Local search is incomplete, but outperforms complete methods by orders of magnitude on satisfiable SAT instances, see, for example, Gu [7] or Selman, Kautz, and Cohen [15].

Building on the success of SAT solvers, first solvers for QBF based on backtracking search have been developed, for example, by Cadoli, Schaerf, Giovanardi, and Giovanardi [2], Giunchiglia, Narrizzano, and Tacchella [6], or Rintanen [13]. Observe that SAT and QBF solvers have the same worst-case behavior: They are both exponential in the number of variables. This does not account for the different complexity of SAT and QBF. As an NP-complete problem, SAT is at the first level of the polynomial hierarchy while QBF is at higher complexity levels. We will focus here on QBF instances at the second level of the polynomial hierarchy which are still considered to be much harder than SAT. For details on the polynomial hierarchy, see Stockmeyer [16]. This subproblem of QBF is called *2QBF* in the following.

Although 2QBF is only a subclass of QBF, there are many important applications that can be represented by 2QBF. For example, Rintanen [13] shows that conformant planning is co-NP^{NP} -hard and translates conditional planning problems to instances of 2QBF. Logic-based abduction, an important problem in automated reasoning, can be modeled by 2QBF as well (Otwell, Remshagen, Truemper [9]).

Local search reflects the difference in complexity between SAT and QBF, respectively 2QBF, better than backtracking search. In fact, local search cannot be applied to QBF or even 2QBF directly. We discuss why local search can solve problems in the class NP—though the resulting solution algorithm is incomplete—but why it cannot solve problems at the second level or at higher levels of the polynomial hierarchy. Nevertheless, local search has a high potential to improve current QBF and 2QBF solvers significantly. We

have developed a complete solution algorithm for 2QBF that uses local search as a heuristic. Based on previous results for local search and 2QBF, we show that our algorithm is a very promising approach and may speed up current 2QBF solvers by orders of magnitude.

2. LOCAL SEARCH FOR SAT

The class NP is typically defined in terms of nondeterministic, polynomial-time Turing Machines. We use an equivalent definition that exposes an important property of problems in NP. Let P be a decision problem. Then the instances of P are divided into “yes” and “no”-instances. For example, the instances of SAT are propositional formulas in *conjunctive normal form (CNF)*, where each formula is a conjunction of *clauses* and each clause is a disjunction of *literals*. A literal is a variable or a negated variable. The “yes”-instances of SAT consist of the CNF formulas that evaluate to *True* under some truth assignment to their variables.

A decision problem is in NP, if there exists a certificate for every “yes”-instance that proves the instance to be a “yes”-instance and that can be verified in polynomial time. See also Papadimitriou [10]. The certificates that show that an instance is a “yes”-instance are called *valid certificate* or *solutions*. In case of SAT, a satisfying truth assignment can be considered to be a solution of a CNF formula. The set of possible certificates comprises all truth assignments to the variables. It can be determined in polynomial time whether a truth assignment satisfies a given CNF formula.

Local search is a solution technique that starts with some solution candidate and attempts to move towards a solution by selecting heuristically a neighboring solution candidate. We outline the local search scheme that has been applied to SAT successfully by [15]. The values *max_tries* and *max_flips* must be given as parameters:

```

For  $i = 1, \dots, \text{max\_tries}$ :
  Select a random solution candidate  $\alpha$ ;
  For  $i = 1, \dots, \text{max\_flips}$ :
    If  $\alpha$  is a solution, return  $\alpha$ ;
    Modify  $\alpha$ ;
Return without a solution;
```

Since local search does not trace the processed solutions α , it cannot explore the search space systematically. Hence, it may or may not terminate with a solution even if one exists. However, local search is *sound*, that is, it never declares an unsatisfiable instance to be satisfiable. In order to avoid situations where, for example, the same set of solutions is iteratively generated, occasionally a new start solution is randomly determined. Local search continues, until a solution has been determined or until a maximum number of iterations has been exceeded. Typically, a heuristic modifies a solution candidate α only slightly in order to produce a neighbor of α which is closer to a solution.

In case of SAT, the candidates α are the possible truth assignments to the variables. The neighboring assignments of α result from flipping a truth value of a single variable. Local search has been shown to be very effective on some NP-complete problems, like SAT (see Gu [7] or Selman, Kautz,

and Cohen [15]). In fact, it has been shown experimentally that local search outperforms the best known complete SAT solvers by orders of magnitude on satisfiable instances. The good performance of local search is contributed to (a) the quality of the heuristics that determines the next truth assignment and (b) the fact that a very large number of assignments can be processed very fast.

We can consider the above local search procedure as a general incomplete solution scheme for problems in NP. The search space consists of all possible certificates. The search procedure guesses certificates and tries to verify them until a valid certificate has been found or until a given number of iterations has been exceeded. In order to ensure efficient execution of each iteration, validation of certificates should not exceed polynomial-time. By definition of NP, polynomial validation time is guaranteed.

We try to apply local search to QBF and its subclass 2QBF of QBF. An instance of 2QBF is a quantified CNF formula of the form

$$\forall X \exists Y T \quad (1)$$

where X and Y partition the variable set of the CNF formula T into two sets. Formula (1) evaluates to *False* if an assignment to the variables in X reduces T to an unsatisfiable CNF formula. It has been shown that 2QBF is co-NP^{NP} -complete and therefore, 2QBF is unlikely contained in NP. That means, we cannot hope to find a certificate for either all *True* instances or for all *False* instances of 2QBF that can be verified in polynomial time. In other words, if local search can be applied to 2QBF where each iteration takes polynomial time, then 2QBF is in NP, which is unlikely. The same result holds for QBF. Hence, local search cannot be employed directly to solve 2QBF or QBF. Nevertheless, we can take advantage of the powerful performance of local search on satisfiable SAT instances. Consider a brute-force solution algorithm that evaluates Formula (1). For every truth assignment to the variables in X , the brute-force algorithm determines whether the resulting formula is satisfiable. Thus, a large number of SAT instances has to be solved in general. In case Formula (1) evaluates to *True*, all SAT instances are satisfiable and thus, local search can possibly solve those instances. We describe a 2QBF solver that takes advantage of this property.

3. LOCAL SEARCH FOR 2QBF

As mentioned before, a brute-force algorithm that evaluates Formula (1) may determine for every truth assignment to the variables in X whether the resulting formula is satisfiable. In order to explore all assignments to X systematically, backtracking search uses a tree, for example. Instead of a tree, Ranjan, Tang, and Malik [12] have suggested to trace all assignments by an additional CNF formula. Each assignment to X corresponds to a clause that evaluates to *False* under that assignments. For example, let $X = \{x_1, x_2, x_3\}$ and $x_1 = \text{True}$, $x_2 = \text{False}$ and $x_3 = \text{True}$ be an assignment to the variables of X . Then the clause $\neg x_1 \vee x_2 \vee \neg x_3$ evaluates to *False*. Under every other truth assignment to X , the clause is *True*. Whenever an assignment to X has been shown to result in satisfiability of T , the respective clause is added to an initially empty CNF formula R . As R is defined

to be the empty CNF formula with variable set X in the beginning, formula R contains no clauses and hence, evaluates to *True* under every truth assignment to its variables. By adding iteratively the clauses to R , we cut off all already processed assignments from the set of satisfying solutions of formula R . In other words, the satisfying solutions of R are the assignments that still have to be checked.

As the number of truth assignments to X is exponential in the size of X , CNF formula R will grow exponentially if a 2QBF instance is *True*. In order to prevent exponential growth of R , we try to determine clauses that cut off more than a single truth assignment. Consider again the previous example with $X = \{x_1, x_2, x_3\}$ where T evaluates to *True* under the assignment $x_1 = \text{True}$, $x_2 = \text{False}$, $x_3 = \text{True}$. Further assume that T evaluates also to *True* under $x_1 = \text{True}$ and $x_2 = \text{False}$ no matter which value is assigned to x_3 . Then the clause $\neg x_1 \vee x_2$ can be added to R since it cuts off exactly all assignments that set x_1 to *True* and x_2 to *False*. The shorter the clause added to R the more assignments can be excluded. We call an assignment to some (possibly all) X variables *Y-satisfiable* if for every possible assignment to the remaining variables in X , the resulting CNF formula is satisfiable. We are interested in a minimal *Y-satisfiable* assignment.

How can we determine a minimal *Y-satisfiable* assignment? We use a heuristic that starts with an *Y-satisfiable* assignment α to all X variables. The heuristic removes tentatively a variable, say x , from assignment α . In addition, All occurrences of the literals x and $\neg x$ are deleted in the clauses in T . The remaining X variables are fixed according to α and the resulting SAT instance is solved. If the instance is satisfiable, variable x is not necessary to satisfy the CNF formula. In case of unsatisfiability, we cannot decide if variable x can be removed from α . Hence, we add x again to the assignment, and the literals x and $\neg x$ are added back to the clauses. Then the next variable is removed tentatively using the same process. The reduced assignment α' is *Y-satisfiable* since the CNF formula does not need any help from the deleted X variables in order to be satisfied. The truth assignment α' is called a *partial Y-satisfying assignment* of α . An analogous process has been used by Otwell, Remshagen and Truemper [9]. Although the heuristic does not guarantee to find a minimal *Y-satisfying* assignment, it has been shown to be very effective.

We summarize the solution algorithm:

Input: CNF formula T with variable set $X \cup Y$ where X and Y are disjoint.

Output: *False*, if there exists an assignment for X that cannot be extended to a satisfying solution of T .
True, otherwise.

1. Set R to be the CNF formula without clauses and with the variable set X .
2. Solve the SAT instance R .
3. If R is unsatisfiable, return *True*.
4. If R is satisfiable, let α be the satisfying truth assignment.

5. Solve the SAT instance resulting from fixing α in T .
6. If T is unsatisfiable, return *False*.
7. If T is satisfiable, determine a partial *Y-satisfiable* assignment α' of α .
8. Add a clause to R that consists exactly of the literals v where v is set to *False* by α' or $\neg v$ is set to *True*.
9. Go to Step 2.

PROOF OF CORRECTNESS. We first show that the algorithm terminates. In each iteration, Step 2 determines an assignment to the variables in X . Since the clause added to R prevents the same assignment to be determined in a succeeding step, R must become unsatisfiable and thus terminate in Step 3 if the algorithm does not terminate before in Step 6.

If the algorithm terminates in Step 6, it has found an assignment to X that results in unsatisfiability of T . Thus, the instance is *False*, and the algorithm returns the correct output. Suppose the algorithm terminates in Step 3. We have to show that every assignment to X is *Y-satisfiable*. Let β be an assignment to X . Since R does not evaluate to *True* under β , there is a clause in R that evaluates to *False* under β . That clause was added previously to R in Step 8. The clause was added since it is unsatisfiable under exactly all assignments that are extensions of a specific partial *Y-satisfiable* assignment. In particular, β is *Y-satisfiable*. \square

The performance of the suggested 2QBF solver depends heavily on the performance of a SAT solver used in Step 2 and 5. Observe that all SAT instances that have to be solved in Step 2 and 5 are satisfiable, save the last instance. Therefore, a local search based SAT solver may solve those instances. Since local search does not guarantee to find a solution even if one exists, we employ a complete SAT solver if local search is unsuccessful. Local search is used as a heuristic that tries to solve a SAT instance before possibly using a complete SAT solver.

In addition to Step 2 and 5, a SAT solver is required to determine partial *Y-satisfiable* assignments in Step 7. Recall that an assignment can be reduced by a variable x if deletion of x in the assignment and in the clauses of the CNF formula results in satisfiability. Local search can be used again to establish satisfiability. Here it is not necessary to employ a complete SAT solver if local search is unsuccessful. In this case, we add the variable again to the assignment and to the clauses. We outline Step 7:

Input: CNF formula T with variable set $X \cup Y$ where X and Y are disjoint; a *Y-satisfiable* assignment α to the variables X .

Output: A partial *Y-satisfiable* assignment of α

For all variables x in X :

Remove all occurrences of x and $\neg x$ from the clauses of T . Let T' be the resulting CNF formula after fixing the remaining variables in X according to α .

Use local search to solve the SAT instance T' .

If local search does not find a satisfying solution, add x again to the clauses of T .

Otherwise, remove x from the assignment α

Return α .

As local search is sound, the resulting assignment is still Y -satisfiable. However, the heuristic in Step 7 may not reduce the assignment with local search to the same extent than it would do with a complete SAT solver.

4. EVALUATION

The performance of the 2QBF solver depends on the following three factors:

- (a) The number of iterations of Step 2 to 9
- (b) The time required by a SAT solver in Step 2 and 5
- (c) The time required to determine a partial Y -satisfying assignment in Step 7

Due to the complexity of 2QBF and SAT, we cannot expect that the number of iterations in (a) is polynomial or that the steps in (b) and (c) take polynomial time. However, we can compare the algorithm with other state-of-the-art QBF and 2QBF solvers.

Our solution scheme for 2QBF is similar to the one developed by Ranjan, Tang, and Malik [12]. To the best of our knowledge, Ranjan, Tang, and Malik [12] have developed and implemented the currently best algorithm specialized to solve 2QBF. There are several excellent QBF solver that can solve 2QBF since 2QBF is a subproblem of QBF. Ranjan, Tang, and Malik [12] have compared the performance of their solver with the performance of current state-of-the-art QBF solvers on 2QBF instances. Their solver is very competitive with QBF solvers. On many problem sets, the 2QBF solver outperforms QBF solvers.

The solvers by Ranjan, Tang, and Malik [12] differs from our algorithm in the SAT solver that is applied in Step 2 and 5. Ranjan, Tang, and Malik [12] employ a complete SAT solver while our solver applies local search as a heuristic first. Empirical studies have shown that local search outperforms SAT solvers by orders of magnitude on satisfiable instances (Gu [7], Selman, Kautz, and Cohen [15]). In fact, instances with thousands of variables that have not yet been solved by complete methods can be solved by local search very efficiently. The local search heuristic in our 2QBF solver is expected to detect satisfiability in the large majority of instances, except for the last instance, which is unsatisfiable. Thus, the runtime should improve significantly. One may argue that the runtime of local search may deteriorate when the CNF formula R has only a few satisfying solutions. However compared with complete SAT solvers, local search is still very competitive in this case.

In addition to the employed SAT solver, we have developed the above described heuristic for Step 7 to reduce Y -satisfiable assignments. The quality of this heuristic is crucial to reducing the number of iterations. Obviously,

the shorter the clause added to R the more assignments are cut off and therefore, the less assignments are left to be processed. Otwell, Remshagen, and Truemper [9] apply the same heuristic in a similar setting where partial Y -satisfiable assignments have to be determined. Experiments show that the truth assignments are reduced substantially. Even though several SAT instances have to be solved in order to reduce a single Y -satisfiable assignment, the heuristic proves to be very fast. We can contribute that to the reduced number of variables of the CNF formula to be solved each time since all X variables are fixed to a truth value or removed from the formula. The heuristic used by Otwell, Remshagen, and Truemper [9] employs a backtracking-search based SAT solver. We suggest here to apply local search instead. The above discussion indicates that local search likely detects satisfiable instances. So the resulting reduction of the Y -satisfiable assignment is expected to be as good as in the case of complete methods.

Although there are many highly efficient local search algorithms available, an implementation of the proposed 2QBF solver requires extensive experimentation with some of the parameter that need to be decided on for local search. For example, the maximum number *max_tries* of tries and maximum number *max_flips* of flips need to be set carefully since all unsatisfiable instances arising in the 2QBF algorithm will process the maximum number of tries and flips. Hence, those parameters have to be set as low as possible but so that satisfiable instances are still solved. We also want to point out that, in case of SAT instances with few variables, local search does not compare so well to the best complete SAT solvers as in the case of very hard large instances. An implementation of our 2QBF algorithm may consider to employ a complete solver if a SAT instance to be solved contains few variables only.

5. RELATED WORK

Our solution algorithm is closely related to the 2QBF solver by Ranjan, Tang, and Malik [12]. They introduce a solver that searches systematically all assignments to the variables in the universal quantifier by adding clauses to a second CNF formula. The heuristic that determines partial Y -satisfiable assignments has been applied by Otwell, Remshagen, and Truemper [9] to logic-based abduction problems. Local search solvers have been developed and implemented, for example, by Gu [7], by Selman, Kautz, and Cohen [15], and by Tompkins and Hoos [3]. The most effective complete SAT solvers are based on backtracking search, like the solver by Bayardo and Schrag [1], by Marques-Silva and Sakallah [8], by Zhang [17], or by Zhang, Madigan, Moskewicz, and Malik [18]).

Recent work on QBF is as follows. Gent, Hoos, Rowley, and Smyth [4] have developed a first QBF solver that combines backtracking search with local search. Their solver is incomplete. Among the most promising complete algorithms for QBF are backtracking search algorithms. Cadoli, Schaerf, Giovanardi, and Giovanardi [2] explore different heuristics to choose the branching literal and investigate properties of randomly generated quantified Boolean formulas. Giunchiglia, Narrizano, and Tacchella [6] implement a learning scheme in their solver QuBE. Rintanen [14] extends the Davis-Putnam procedure by unit propagation for

quantified formulas. The QBF solver Quaffle by Zhang and Malik [19] learns conflict clauses during the solution process. The solver QSAT_{CNF} by Plaisted, Biere, and Zhu [11] is based on a DPLL procedure and uses the SAT solver SATO (Zhang [17]).

6. CONCLUSIONS

Although local search cannot be employed directly to QBF or to 2QBF, we have described a very promising 2QBF solver that combines a newly proposed algorithm for 2QBF with local search. In contrast to local search based SAT solvers, our solution algorithm is complete. Employing highly efficient local search algorithms is expected to improve the runtime of state-of-the-art 2QBF solvers by orders of magnitude. We believe that this result deserves further research. We are currently implementing the proposed 2QBF solver to experiment with the various possible parameter settings for local search. Instead of using the same settings for all instances, we expect that different applications of 2QBF may perform best with individual parameters.

7. REFERENCES

- [1] R. J. Bayardo Jr. and R. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 203–208, 1997.
- [2] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An Algorithm to Evaluate Quantified Boolean Formulae and its Experimental Results. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
- [3] D. A. D. Tompkins and H. Hoos. UbcSAT: An implementation and experimentation environment for SAT algorithms for SAT and MAX-SAT. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, BC, Canada, May 2004.
- [4] I. P. Gent, H. Hoos, A. G. D. Rowley, and K. Smyth. Using stochastic local search to solve quantified boolean formulae. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-03)*, pages 348–362, 2004.
- [5] E. Giunchiglia, M. Narizzano, and A. Tacchella. QBF reasoning on real world instances. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, BC, Canada, May 2004.
- [6] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artificial Intelligence*, 145(1):99–120, 2003.
- [7] J. Gu. Efficient local search for very large scale satisfiability problems. *Sigart Bulletin*, 3(1):8–12, 1992.
- [8] J. P. Marques-Silva and K. A. Sakallah. GRASP—A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.
- [9] C. Otwell, A. Remshagen, and K. Truemper. An effective qbf solver for planning problems. In *Proceedings of the International Conference on Algorithmic Mathematics and Computer Science*, pages 311–316, June 2004.
- [10] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994.
- [11] D. A. Plaisted, A. Biere, and Y. Zhu. A satisfiability procedure for quantified boolean formulae. *Discrete Applied Mathematics*, 130:291–328, 2003.
- [12] D. Ranjan, D. Tang, and S. Malik. A comparative study of 2qbf algorithms. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, BC, Canada, May 2004.
- [13] J. Rintanen. Constructing conditional plans by a theorem prover. *ACM Trans. Program. Lang. Syst.*, 10:323–352, 1999.
- [14] J. Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified boolean formula. In *International Conference on Logic Programming, Artificial Intelligence and Reasoning*, pages 362–376, 2001.
- [15] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*, October 1993.
- [16] L. J. Stockmeyer. The polynomial hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.
- [17] H. Zhang. SATO: an Efficient Propositional Prover. In *Proceedings of the International Conference on Automated Deduction (CADE-97)*, pages 272–275, 1997.
- [18] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design*, 2001.
- [19] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design*, 2002.