Finding small unsatisfiable cores to prove unsatisfiability of QBFs

Yannet Interian[1] Gabriel Corvera[2] Bart Selman[3] Ryan Williams[4]

[1] *Center for Applied Mathematics. Cornell University, Ithaca, NY 14853*

interian@cam.cornell.edu

[2] *Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Estado de México, and Department of Computer Science, Cornell University*

gec27@cs.cornell.edu

[3] *Department of Computer Science, Cornell University, Ithaca, NY 14853*

selman@cs.cornell.edu

[4] *Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213*

ryanw@cs.cmu.edu

Abstract

In the past few years, we have seen significant progress in the area of Boolean satisfiability (SAT) solving and its applications. More recently, new efforts have focused on developing solvers for Quantified Boolean Formulas (QBFs). Recent QBF evaluation results show that developing practical QBF solvers is more challenging than one might expect. Even relatively small QBF problems are sometimes beyond the reach of current QBF solvers. We present a new approach for solving unsatisfiable two-alternation QBFs. Our approach is able to solve hard random QBF formulas that current algorithms are not able to handle.

Our solver WalkMinQBF combines the power of stochastic local search methods and complete SAT solvers. The solver is incomplete, in that it outputs *unsat* if a certificate for unsatisfiability is found, otherwise it outputs *unknown*. We test our solver on the model for random formulas introduced in [3] and the Models A and B introduced in [7]. We compare WalkMinQBF with the state-of-the-art QBF solvers `Ssolve` and `QuBE-BJ`. We show that WalkMinQBF outperforms `Ssolve` and `QuBE-BJ` in time and in the number of formulas solved. We believe our work provides new insights on strategies that should be useful in complete QBF solvers. As a side result we have developed a stochastic local search algorithm for the minimum unsatisfiability problem (MIN-SAT).

1 Introduction

Recently, we have seen tremendous progress in the performance of Boolean satisfiability (SAT) solvers. Despite the worst-case exponential runtime of all known algorithms, SAT solvers are now in everyday use for many applications. Encouraged by the tremendous advances in boolean satisfiability, many researchers have recently turned to studying a powerful generalization of boolean satisfiability: the quantified boolean formula (QBF) problem. The QBF problem allows for the modeling of problems having higher complexity than SAT, including problems from the areas of verification, planning, knowledge representation, game playing, logic, and combinatorics. Recall that SAT is known to be NP-complete, and QBF is known to be PSPACE-complete. Recent QBF evaluation results [11] show that developing practical QBF solvers is more challenging than one might expect. Even relatively small QBF problems are sometimes beyond the reach of current QBF solvers.

The need for practical QBF solvers presents new challenges on deriving heuristics to prune parts of the search space. We present here an approach for solving unsatisfiable two alternation QBFs. Our solver is based on heuristics that effectively find inconsistencies for certain unsatisfiable formulas. We believe that our work gives insights on what type of heuristics complete QBF solvers have to consider.

Our new approach, WalkMinQBF, combines the power of stochastic local search methods and complete SAT solvers. The solver we present is incomplete. More precisely, it outputs *unsat* if a certificate for unsatisfiability is found otherwise it outputs *unknown*. WalkMinQBF has two main components. The first one is a local search algorithm that attempts to find an assignment to the universal variables that is a witness for unsatisfiability. The second one is a complete SAT solver (`kcnfs` [4]) that tests for TRUE/FALSE of SAT formulas that result from assigning the universal variables. The algorithm iteratively finds universal assignments to "feed" the SAT solver until an unsatisfiable subformula is found or the maximum number of iterations is reached.

To show the effectiveness of our approach, we use the model for random formulas introduced in [3]. Although our discussion concentrates on random formulas from the model in [3], we also present experiments with instances from *Model A* and *Model B* introduced in [7]. Random formulas are a good choice for at least two reasons. First, the random models described in [3] and [7] have a "phase transition" property, which allow us to find hard unsatisfiable instances by looking at values of the parameter near the threshold. Second and most important, we have a homogeneous test bed with challenges for proving unsatisfiability. We believe that the type of difficulties involved in proving unsatisfiability varies greatly among different QBF domains. In particular, we have found that for a hard random unsatisfiable formula, there are often just a few assignments to the universal variables that lead to an unsatisfiable subformula. We call such a subformula an *unsatisfiable core*. Usual DPLL-based algorithms find these random instances to be quite difficult. These algorithms lack heuristics that would detect these inconsistencies, so they typically search most of the space to solve these instances.

We compare WalkMinQBF with the state-of-the-art QBF solvers `Ssolve` [5] and `QuBE-BJ` [8]. We show that WalkMinQBF outperforms `Ssolve` and `QuBE-BJ` in time and in the number of formulas solved. `Ssolve` is the solver that performed the best for the 2004 and 2005 QBF evaluation in the category of random formulas [11]. `QuBE-BJ` performed better than `Ssolve` on random instances on the first stage of the 2004 evaluation, but was not considered for the second stage.

Along the way, we develop a stochastic local search algorithm walkMinSAT for the minimum unsatisfiability problem (MIN-SAT). The algorithm is in the same spirit as WalkSat [10], a highly successful local search algorithm for the Satisfiability (SAT) problem and Maximum Satisfiability problem (MAX-SAT). The algorithm starts at a random assignment, at each step, chooses b different "weakly" satisfied clauses, that is clauses with one satisfied literal, and flips one of these satisfied variables. walkMinSAT is used together with a "balance" heuristic for sampling the space of universal assignments (equivalently, the corresponding space of subformulas obtained by instantiating these assignments) to try to find one unsatisfiable subformula.

The remainder of this paper is organized as follows. Background on QBF, the QBF random model, the instances used for testing the algorithm and the MIN-SAT problem are discussed in the next Section. In Section 3, we provide a detailed discussion of the algorithm and its parts. Experimental results and discussion are given in Section 4. Finally, in Section 5 we summarize our work and discuss some open questions.

2 Preliminaries

Formulation of QBF. In this paper, we use the formulation of QBF where consecutive variables having the same quantifier are combined into a set. The following are the general forms of QBF formulas having one, two, and three *quantifier blocks*, respectively:

$$\exists X_1 \phi, \quad \forall Y_1 \exists X_1 \phi, \quad \exists X_2 \forall Y_1 \exists X_1 \phi$$

By a *quantifier block*, we mean an expression QV where Q is a quantifier ($\exists$ or $\forall$) and V is a set of boolean variables. In general, an instance of QBF contains a sequence of quantifier blocks which alternate between universal and existential type, followed by a set of clauses, denoted here by ϕ.[1]

The semantics of these formulas are defined as is standard in logic; for instance, a formula $\forall Y_1 \exists X_1 \phi$ is true if for every assignment to the variables Y_1, there exists an assignment to the variables X_1 such that ϕ is true. An intuitive way to view a QBF formula is as a game between two players: a "universal player", who attempts to falsify ϕ, and an "existential player", who attempts to satisfy ϕ. Players set their variables in the order given by the quantifier prefix. For instance, a formula $\exists X_2 \forall Y_1 \exists X_1 \phi$ can be viewed as a three-round game in which the existential player first sets the variables X_2, the universal player then sets the variables Y_1, and the existential player finishes by setting the variables X_1. In general, the formula is true if the existential player can always win the game by satisfying ϕ.

2.1 Random SAT and random QBF models.

Let us review the standard random SAT model, and then a generalization: a random QBF model. The standard SAT model [9] takes three parameters: the clause length k, the number of variables n, and the number of clauses C. A random formula with these parameters, denoted here by k-$\mathcal{F}(n, C)$, is generated one clause at a time; each clause is generated by selecting, uniformly at random, k distinct variables, and then negating each with probability one half.

Now we describe the random QBF model introduced in [3]. As in the random SAT model, every clause has the same length in the QBF model. In addition, the number of literals in a clause from each quantifier block is also constant. The QBF random model has three parameters. The first parameter of the model is a tuple specifying, for each quantifier block, the number of literals from that block that are in a clause. For instance, the tuple $(2, 3)$ specifies that the generated formulas are of the form $\forall Y_1 \exists X_1 \phi$, where each clause in ϕ has 5 literals, 2 from Y_1 and 3 from X_1. These formulas are denoted $(2, 3)$-QBF formulas. As another example, the tuple $(4, 2, 3)$ specifies that the generated formulas are of the form $\exists X_2 \forall Y_1 \exists X_1 \phi$, where each clause in ϕ has 9 literals, 4 from X_2, 2 from Y_1, and 3 from X_1. Analogously, we call such formulas $(4, 2, 3)$-QBF formulas.

The second parameter of the model is a tuple of the same length as the first one, specifying the number of variables in each quantifier block. The third parameter to our model is simply the number of clauses C. As an example, suppose that the first tuple is $(2, 3)$, the second tuple is $(64, 80)$, and the number of clauses is 840. The generated formulas are of the form $\forall Y_1 \exists X_1 \phi$, where Y_1 has 64 variables, X_1 has 80 variables, each clause in ϕ has 2 literals from Y_1 and 3 literals from X_1, and there is a total of 840 clauses in ϕ. Each clause is generated by selecting uniformly at random, for each quantifier block, the designated number of distinct variables, and then negating each variable with probability one half. We denote such a random formula by $(2, 3)$-$\mathcal{F}((64, 80), 840)$.

Note that in a random QBF from this model, if any quantifier block is picked and the clauses are restricted to literals over the quantifier block, the result is a random formula over the variables

[1] Note that we use the standard assumption that the innermost quantifier block is existential, for if it is universal, the block can be efficiently eliminated. We also assume, as usual, that different quantifier blocks have disjoint variable sets.

of the quantifier block in the sense of the random SAT model. Instantiating the variables from the outermost quantifier block "without looking at the rest of the formula" also results in another random QBF from the model, under the appropriate choice of number of clauses.

In this paper, we discuss two-alternation formulas, *i.e.* formulas of the form $F = \forall Y \exists X \phi$. Denote by F_Y the SAT formula obtained by considering from each clause only the literals from the Y variables, we call this formula the "universal formula" or the "Y-formula". In addition we define the "existential formula" or "X-formula" to be the SAT formula obtained from F by considering in each clause just the existential literals, we denote it by F_X. We define a third SAT formula associated with F, as a subformula of F_X obtained by instantiating the Y variables and considering the SAT subformula that remains. We denote this subformula by $F_{y,X}$ where y is an assignment to the variables in Y.

In [3] the phase transition property of the QBF model is discussed. The paper provides experimental results that show that when fixing the number of variables while increasing the number of clauses, the formulas change from being almost certainly true to almost certainly false. This transition seems to take place relatively abruptly, suggesting that the model exhibits a threshold phenomenon as conjectured in the random SAT model. Furthermore, we conjecture that if we take the number of existential variables as the "parameter that goes to infinity" we have the right parametrization in the following sense:

Conjecture: If we fix the parameters ρ, k, l, where ρ is the ratio between the universal and existential variables, then there exist c_* such that if $F \in (k,l)\text{-}\mathcal{F}((\rho n, n), cn)$ and $c < c_*$, F is satisfiable *with high probability*[2] (w.h.p.) and if $c > c_*$ then F is unsatisfiable w.h.p.

For instance experiments in [3] show that for $\rho = 0.8$ for $(2,3)$-QBF formulas the value of c_* is around 9. Therefore we can use this model to generate unsatisfiable formulas to test our algorithm. Moreover, we expect that the most difficult formulas are the ones with c around the value c_*.

Hard random $(2,3)$-QBF: Test instances. To evaluate our approach, we run experiments on the $(2,3)$-QBF random formulas. Experimental results in [3] showed that the hardest formulas were found when the ratio between universal and existential variables is around 0.8. Therefore, we fixed $\rho = 0.8$ for our experiments. We generate formulas with parameters $c = 10.0, 10.3, 10.5$ and $n = 80$. Note from the discussion above that the formulas at $c = 10.0$ are the hardest ones. We also use the parameters $c = 10.5$ and $n = 80, 100, 200$, that corresponds to $144, 180, 360$ total number of variables respectively.

These formulas are experimentally very hard. For example, consider $F \in (2,3)\text{-}\mathcal{F}((0.8n, n), 10.5n)$. Note that, if y is a random assignment to the universal variables, it satisfies each clause with probability $3/4$ therefore $F_{y,X}$ has around $1/4$ of the original number of clauses. Moreover, $F_{y,X}$ is a random 3-SAT formula with density approximately 2.625. Such formulas are always satisfiable. Thus, we are considering formulas in which most of the y assignments lead to SAT subformulas but still the QBF formula in unsatisfiable. That is, there are only a few assignments to the universals that lead to unsatisfiable subformulas. The most natural strategy to consider is to find y assignments that satisfy the least number of possible clauses. Experimentally, we have found that $\max_y \#\text{clauses}(F_{y,X}) \approx 3.8n$, *i.e.* the subformulas with the maximum number of clauses have density 3.8. Still, these subformulas with ratio 3.8 are typically satisfiable but a very small fraction of them are unsatisfiable (more discussion in the next section). So, we have to find a hidden small unsatisfiable subformula of ratio approximately 3.8. We call these subformulas *small unsatisfiable cores.*

[2] With probability that goes to 1 as the n goes to infinity.

Other QBF models. In this paper, we mainly focus on the QBF model described above, but we also ran experiments with other QBF models. In [7], two models for random QBF were defined. Formulas for *model A* are generated in a similar way as a random k-SAT formula. Each clause of fixed length k is generated by randomly choosing k distinct variables from the entire set of variables, and then negating each with probability one half. Clauses with less than j existential variables are discarded and replaced, where j is a parameter with values $j \geq 2$. *Model B* is generated in a similar way as *model A* but each clause has exactly $j \geq 2$ existential variables. *Model B* can be obtained form (l, j)-$\mathcal{F}((\rho n, n), cn)$ by taking $\rho = 1$.

MIN-SAT problem. Let F be a SAT formula on n variables. Define $\mathsf{minSAT}(F)$ to be the minimum, over all 2^n assignments, of the number of clauses satisfied in the formula F.

3 WalkMinQBF Algorithm

Let F be a two-alternation formula $\forall Y \exists X \phi$. In order to prove that such a formula is unsatisfiable, we have to find an assignment y to the universal variables Y such that the SAT formula $F_{y,X}$ that remains after instantiating the assignment is unsatisfiable. To approach this problem we use a local search algorithm on the space of the universal Y-variables to try to find an assignment that corresponds to an unsatisfiable subformula. A first naive idea is to assign random values to the Y-variables and solve the corresponding subproblem with a SAT solver. As we discussed in the previous section this is an ineffective approach. We ran experiments for two hours on 50 random problems with 144 variables and 840 clauses ($(2, 3)$-$\mathcal{F}((64, 80), 840)$) and did not find solutions for any of the instances. The second natural idea is to consider Y-variables that minimize the number of clauses that are satisfied, the corresponding remaining SAT subformula has a larger number of clauses and therefore has more possibilities of being unsatisfiable. We discuss this approach in the next subsection.

3.1 MIN-SAT Algorithm: walkMinSAT

In this subsection we consider a local search algorithm for the minimum satisfiability problem (MIN-SAT). Our algorithm follows some of the intuitions and strategies developed in WalkSat for solving SAT and MAX-SAT problems [10]. We use this algorithm to find assignments to the universal variables that minimize the number of satisfiable clauses in the two-alternation QBF formula.

We start by describing the WalkSat algorithm. The strategy BEST for WalkSat does as follows. It starts from a random assignment. At each step, it picks a random unsatisfied clause $\mathcal{C}$ and picks a variable from that clause in the following way. Define the "break value for a variable" to be the total number of clauses that are true in the current state, but that would become false if the flip were made. If there is a variable in $\mathcal{C}$ with break value 0, flip that variable. Otherwise with probability p, the heuristic takes a literal at random from that clause and flips it. With probability $1 - p$, it flips the variable with smallest "break value". The algorithm finishes when a satisfied assignment has been found or when the number of flips done is equal to the cutoff parameter.

Our algorithm for MIN-SAT (walkMinSAT) tries to follow a similar method. It runs for a maximum number of steps (cutoff). The algorithm outputs the assignment with the least number of satisfied clauses found. It starts from a random assignment. At each step, the algorithm selects at random b different clauses with exactly one satisfied literal. The variable to flip is chosen from the variables corresponding to the satisfied literals by a procedure called **ChooseVar**. The basic algorithm is describe in Figure 1.

```
walkMinSAT($F$,$b$, cutoff,**ChooseVar, score**)
for $i = 1$ to cutoff
    for $j = 1$ to $b$
        $c_j$:= a clause with one satisfied literal chosen at random
        $v_j$:= variable corresponding to the satisfied literal in $c_j$
    $x$:= a variable chosen by heuristic **ChooseVar**($v_1, \ldots, v_b$)
    flip the value of variable $x$
**return** the assignment with the least number of satisfied clauses or best **score**.
```

Figure 1: Random walk strategies for minSAT

The variable to be flipped can be chosen in different ways. We describe a strategy that resembles the BEST strategy for WalkSat. We called it the WORST, since this strategy tries to *minimize* the number of satisfied clauses, contrary to WalkSAT. With probability p, the heuristic just flips one variable at random from the set $L = \{v_1, \ldots, v_b\}$ calculated as in Figure 1. With probability $1-p$ it chooses the variable from L that minimizes the total number of clauses that are false in the current state, but that would become TRUE if the flip were made (*make value*) The procedure for choosing the variable to flip is describe in Figure 2.

```
**ChooseVar**($v_1, \ldots, v_b$)
    $m_j$:=total number of clauses that are false in the current state,
    but that would become true if $v_j$ were flipped, for $j \in \{1 \ldots b\}$.
    **with probability** $p$
        returns variable with minimum $m_j$
    **with probability** $1-p$
        returns variable $v_1$
```

Figure 2: Heuristic WORST for MIN-SAT

3.2 Experiments with walkMinSAT

The first two plots in Figure 3(a) compare the basic algorithm form Figure 1 using the heuristic from Figure 2 with two different values of p and $b = 10$. In Figure 3(a), RANDOM refers to the heuristic in Figure 2 for $p = 0$, *i.e.* taking at random a clause with one satisfied literal and flipping the variable corresponding to the true literal. WORST refers to the heuristic from Figure 2 with $p = 1$. The goal of these experiments was to show that even the basic idea of flipping variables for weakly satisfied clauses is a good strategy. Note that, even the heuristic RANDOM solves 18 out of 50 formulas. The heuristic WORST, which flips variables in a more greedy fashion solves 23 out of 50 and does so faster than RANDOM.

Figure 3(b) plots the distribution of ratios (density) of the remaining 3-SAT subformulas found by RANDOM and WORST. Note that RANDOM did not get very close to the best values, while WORST most of the time got to the best values. For this particular instance, most of the time WORST found subformulas at ratios near 3.8.

In order to apply the MIN-SAT algorithm for solving QBFs we need another ingredient. It is not enough to be able to find the best solution for the MIN-SAT problem. We have to be able to explore many different solutions that correspond to different subformulas. Better than just finding a good solution, we have to sample the space of suboptimal solutions. We deal with this problem at the moment by using very low values of the cutoff (=2000) so that we get different suboptimal solutions. In general, one would like to have a more robust approach.

As previous experiments show, considering Y-assignments with a low number of satisfied clauses increases our chances of getting an unsatisfiable subformula. Nevertheless, for difficult problems, these chances are fairly low. Figure 3(c) shows that walkMinSAT with heuristic WORST usually finds subformulas with very similar ratios. This leads us to consider the following experiment. We generate a random 3-SAT formula at ratio 10.5 with 80 variables and consider random subformulas with densities $c = 3.5, 3.6, 3.7, 3.8$. If Figure 3(c) we consider the fraction of unsatisfiable subformulas from 1,000,000 independent experiments. First note that at these ratios the number of unsatisfiable subformulas is very low. Second, even if low, the fraction increases in an exponential way as a function of the density c. This experiment suggests how important it is to try to find assignments to the universal variables that satisfy the fewest clauses possible.

Until now, our heuristics have been trying to get the least number of satisfied clauses. The choices of the universal assignments were not exploiting any information about the existential formula. The question we address in the next section is how to sample the space of subformulas (corresponding to Y-assignments) in a way to increase the chances of getting an unsatisfiable subformula by using more information about the existential subformula.

3.3 Balance

In the previous subsection we discussed a MIN-SAT algorithm and how it can be used to find assignments to the universal variables to prove unsatisfiability of the two-alternation QBF formulas. We have shown how considering assignments to the universal variables that minimize the number of satisfiable clauses gives subformulas with higher density that are more likely to be unsatisfiable. In this section, we refine MIN-SAT heuristic to be able to use more information on the remaining subformulas by introducing a measure of "balance". We show that subformulas with lower "balance" values are more likely to be unsatisfiable. We use the "balance" measure to derive heuristics combined with MIN-SAT are more successful finding unsatisfiable subformulas.

To understand our new challenge, consider again a formula $F \in (2,3)\text{-}\mathcal{F}((64,80),840)$. When we instantiate an assignment to the Y variables (that does not look at the existential formula) we get a random 3-SAT formula with $n = 80$ variables. These formulas have the same distribution as those of the experiment in the previous section, given the number of satisfied clauses. That is, we are looking at subformulas of a random 3-SAT formula with a small ratio. These subformulas are usually satisfiable. As we saw in our experiment in Figure 3(c) only a very small fraction of them are unsatisfiable. The key question is: how to recognize the unsatisfiable ones without running the subformulas with a SAT solver? Perhaps a more tractable question is: how to sample from a subspace of subformulas to increase our chances of finding the unsatisfiable ones?

Our insights come from the results in [1]. The authors discuss a model for random formulas in which all variables have almost the same number of occurrences. Furthermore, the formulas are constrained to be balanced in the following sense. For every variable x, the number of occurrences of x and $\bar{x}$ differ at most by one. This model exhibits a similar threshold property than the usual model for random k-SAT. Experiments with the regular model show that for $c < 3.5$, almost all formulas are satisfiable, while for $c > 3.5$ almost all formulas are unsatisfiable. Note that the threshold parameter $c_* = 3.5$ is much smaller than the same parameter for the usual 3-SAT model [9], which has been empirically shown to be approximately 4.25. We believe that the shifting of the unsatisfiability threshold is caused by the fact that the formulas are constrained to be balanced.

The later observation suggests that trying to sample the space of a more "balanced" subset of subformulas could help speed up our search of finding unsatisfiable subformulas. We define the

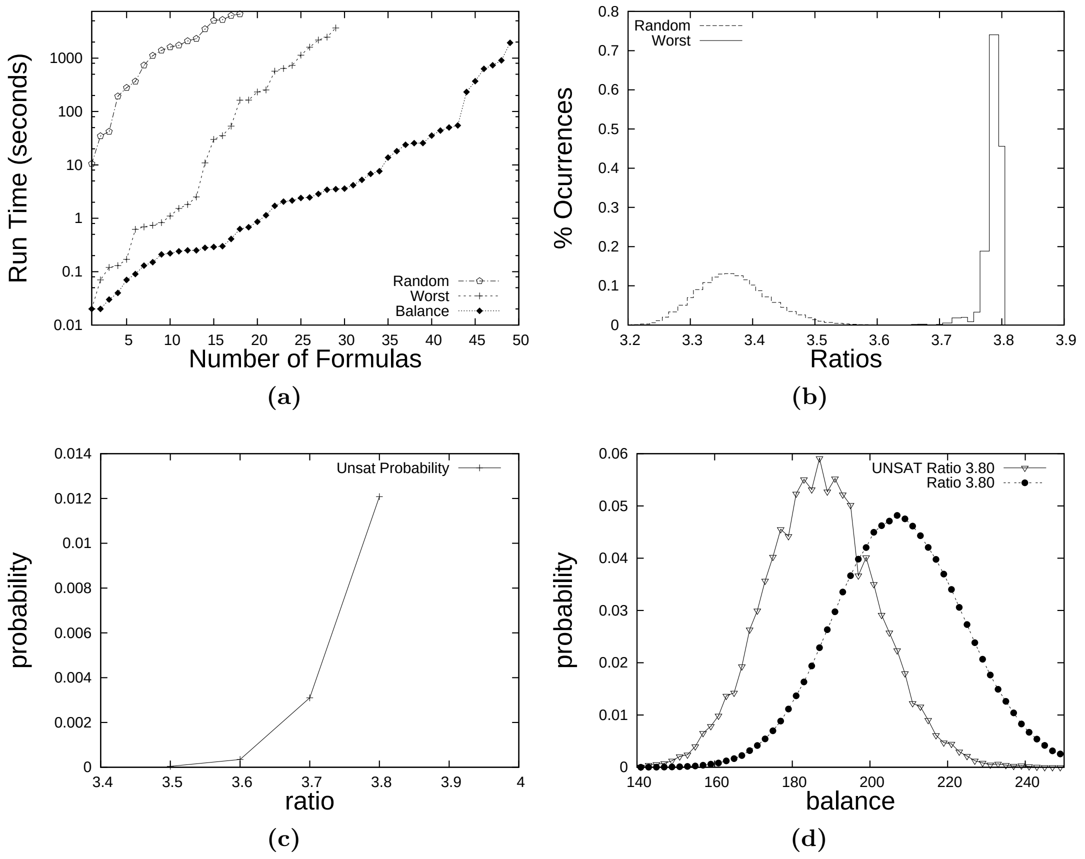


Figure 3: **(a)** Comparison between the three heuristics: RANDOM, WORST, BALANCE on 50 random formulas with 144 variables. **(b)** Comparison between RANDOM, WORST on the distribution of ratios on the remaining 3−SAT formula. **(c)** Probability that a random subformula of a 3-SAT formula is unsatisfiable as a function of the clause density of the subformula. Statistics done on 1,000,000 subformulas of a random formula from 3-$\mathcal{F}(80, 840)$. **(d)** Distribution of the balance function on 1,000,000 subformulas at ratio 3.8 of a formula from 3-$\mathcal{F}(80, 840)$, and the same distribution just on the unsatisfiable subformulas

balance function for a SAT formula F to be

$$\texttt{balance}(F) = \sum_{x \in V} |\texttt{occ}(x) - \texttt{occ}(\bar{x})|$$

where $\texttt{occ}(x)$ is the number of clause in which the literal x occurs in F. Our heuristic leads the search toward subformulas with smaller balance values. We identify the balance of an assignment to the universal variables $\texttt{balance}(y)$ with $\texttt{balance}(F_{y,X})$. The balance value of formulas from the random model defined in [1] is very small, less than the number of variables. More precisely, $|\texttt{occ}(x) - \texttt{occ}(\bar{x})| = 0$ or 1 for every variable x.

To transform the intuition of balance into a strategy, we consider the heuristic in Figure 4. The heuristic BALANCE chooses variables to simultaneously decrease the balance function and decrease the number of clauses that are satisfied. It takes the variables from the list L given by the Walk-MinQBF algorithm described in Figure 5. The balance heuristics first sorts the variables from L to find the one with the best two "m_j" values. From these variables, it outputs the variable which, if

```
**ChooseVar-Balance**$(v_1, \ldots, v_k)$
    $L := \{v_1, \ldots, v_k\}$
    $m_j$:= *make value*:=total number of clauses that are false in the current,
    state but that would become true if $v_j$ were flipped for $j \in \{1 \ldots k\}$.
    $M :=\ \min_j m_j$
    with probability $p$
        **return** a variable with *make value* $M$
    with probability $1 - p$
        if not all $m_j$ are equal let $M' < M$ be the second minimum value.
        Consider a new list $L' := \{v_{j_1} \ldots v_{j_l}\} \subset L$ with all the variables
        with $m_j$ values equal to $M$ or $M'$
        **return** a variable from $L'$ that, if flipped, gives the best balance value.
```

Figure 4: Heuristic BALANCE

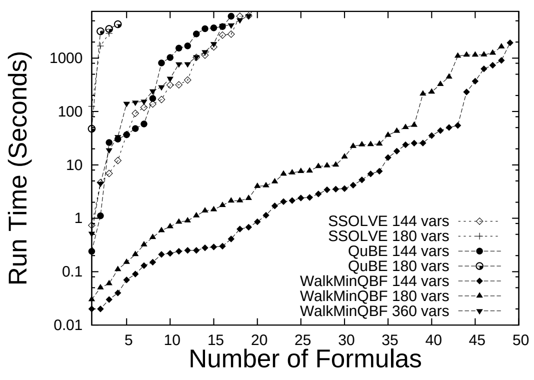
flipped, has the best balance value.

```
WalkMinQBF($F$,$k$, cutoff,MAX-TRIES)
    score($y$):=number of satisfied clauses $+ 0.5 \cdot balance(y)$
    for $i = 1$ to MAX-TRIES
        $y$ = walkMinSAT($F$,$k$, cutoff,**ChooseVar-Balance, score**)
        $F' = F_{y,X}$
        if (kcnfs($F'$) == UNSAT)
            **return** UNSAT
    **return** UNKNOWN
```

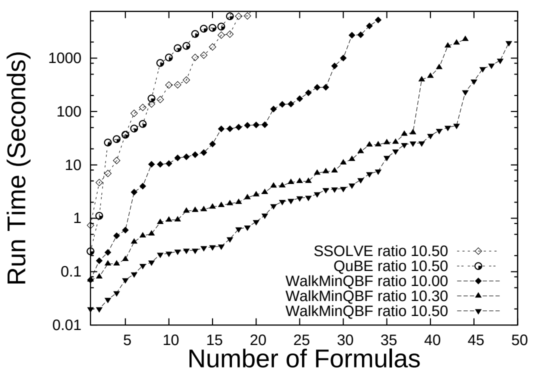
Figure 5: WalkMinQBF: QBF algorithm

To support the choice of the balance function, we ran an experiment similar to the one in the previous section. We generate 1,000,000 random subformulas drawn from 3-$\mathcal{F}(80, 840)$, with density 3.8 (*i.e.* 304 clauses), and for each subformula we calculate its balance value. We also ran the SAT solver on each subformula to find out if it is satisfiable or unsatisfiable. The results of this experiment are given in Figure 3(d). The second plot, called "ratio 3.80", is the frequency of each value of the balance function. The first plot called "UNSAT ratio 3.80" is the same frequency, except just for the unsatisfiable formulas. Note that the distribution of the unsatisfiable formulas is shifted to the left, *i.e.*, the unsatisfiable formulas tend to have smaller balance values.

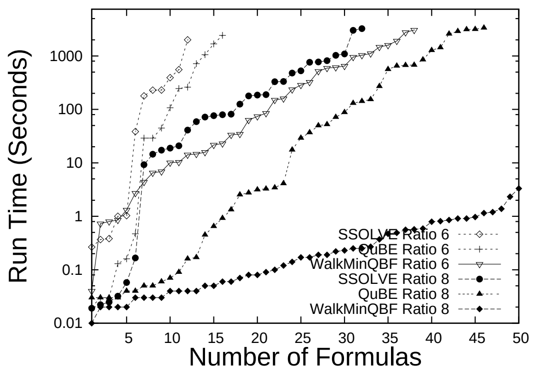
To better understand the plots, let us look at some statistics. From the 1,000,000 total subformulas at ratio 3.8, only 12079 are unsatisfiable. If we were trying to find one of these unsatisfiable formulas by uniformly sampling from that space, the probability of finding one would be 0.012. Now, if we were sampling from formulas with balance < 200 that probability increases to 0.03. If we go further and sample from subformulas with balance < 180, the probability increases to 0.078. The key feature of our algorithm is to shift the sampling to subformulas with smaller balance values. The third plot in Figure 3(c) compares the BALANCE heuristic with previously discussed heuristics: WORST and RANDOM.



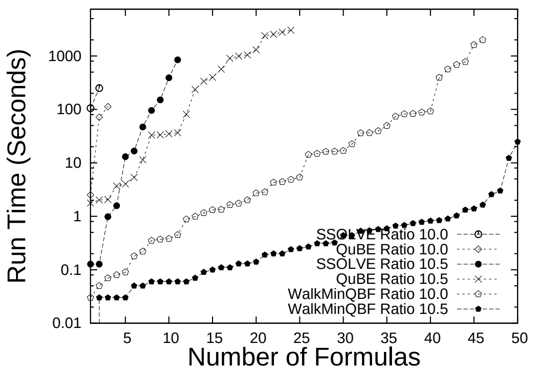
(a)



(b)



(c)



(d)

Figure 6: **(a)** Running time in seconds of WalkMinQBF, `QuBE-BJ`, and `Ssolve` on 50 random formulas from $(2,3)$-$\mathcal{F}((0.8n,n),10.5n)$ for different values of $n = 80, 100, 200$. **(b)** Running time in seconds of WalkMinQBF, `QuBE-BJ`, and `Ssolve` on 50 random $(2,3)$-$\mathcal{F}((64,80),c80)$ for different values of $c = 10, 10.3, 10.5$. **(c)** Running time in seconds of WalkMinQBF, `QuBE-BJ`, and `Ssolve` on 50 random formulas from *Model A*, 150 variables and ratios $c = 6, 8$. **(d)** Running time in seconds of WalkMinQBF, `QuBE-BJ`, and `Ssolve` on 50 random formulas from *Model B*, 100 variables and ratios $c = 10, 10.5$.

4 Experiments with WalkMinQBF and discussion

Most of the current state-of-the-art solvers are based on an extension of the DPLL algorithm, first proposed in [2]. Moreover, most of the current solvers for QBF are complete. An exception is WalkQSat [6], an incomplete solver developed to solve mainly SAT instances that uses WalkSat algorithm together with a QBF solver.

Our test problems are very difficult for current QBF solvers. As we discussed before, just a few assignments to the y variables can prove the unsatisfiability of such formulas. DPLL style solvers have to search most of the space before finding these inconsistencies. `Ssolve` solved some of these instances thanks to its "adaptive trivial falsity" heuristic. For the two alternation formulas the heuristic tries to find in a greedy way some "set of conflict free clauses" of maximal size; that is, to find assignments to the universal variables satisfying the least number of clauses. Most other solvers are not able to handle these problems. For instance, Quaffle, another state-of-the-art QBF solver was not able to find solutions for any of the 50 problems from $(2,3)$-$\mathcal{F}((80,64),840)$ within 2 hours (each problem).

We compare WalkMinQBF with the state-of-the-art QBF solvers `Ssolve` [5] and `QuBE-BJ` [8]. We show that WalkMinQBF outperforms `Ssolve` and `QuBE-BJ` in time and in the number of formulas solved. `Ssolve` is the solver that performed the best for the 2004 and 2005 QBF evaluation in the category of random formulas [11]. `QuBE-BJ` performed better than `Ssolve` on random instances on the first stage of the 2004 evaluation but was not considered for the second stage.

In Figures 6 (a) and (b), we plot the running time (max 2 hours) versus the number of formulas solved by `Ssolve`, `QuBE-BJ`, and WalkMinQBF, for formulas drawn from the $(2,3)$-QBF random model.

Figure 6(a) shows experiments in which the parameter $c = 10.5$ is fixed and let the number of total variables be $144, 180, 360$. We ran `Ssolve` and `QuBE-BJ` for number of variables $vars = 144, 180$, for formulas with 144 variables `Ssolve` solved 24 out of 50 formulas and `QuBE-BJ` solved 20, for ones with $vars = 180$ variables `Ssolve` just solved 3 formulas and `QuBE-BJ` solved 4. WalkMinQBF solved 49 formulas for $vars = 144$, 48 formulas for $vars = 180$ and 19 for $vars = 360$.

Figure 6(b) shows experiments in which the number of variables is fixed $vars = 144$ and we vary c to $c = 10, 10.3, 10.5$. For the most difficult formulas at ratio $c = 10$, WalkMinQBF solved 34 of these instances.

Figure 6(c) and (d) shows the results of running the three solvers on formulas from *Model A* and *Model B* respectively for a maximum of one hour (each formula). For *Model A* we used 5-QBF formulas (each clause has 5 literals) with 150 variables and clause ratios $c = 6, 8$. We set $j = 2$, that is all clauses with less than 2 existential literals were discarded and replaced. For *Model B*, we used 100 variables $c = 10, 10.5$ and $j = 3$, that is, all clauses have exactly 3 existential variables and 2 universal variables. Note that these instances can be obtained also from the random (2,3)-QBF model by using $\rho = 1.0$. We are using $j = 3$ in *Model B*, because when $j = 2$ the three solvers found the instances to be very easy. This is probably due to the fact that the existential formula is a 2-SAT formula.

5 Conclusions and future work

We presented WalkMinQBF, an algorithm to prove unsatisfiability of two-alternation QBF formulas. The algorithm combines the power of stochastic local search methods and complete SAT solvers. We have also developed a local search algorithm for the MIN-SAT problem. We tested WalkMinQBF on hard random instances showing that WalkMinQBF outperforms current solvers on these problems.

Most DPLL-based QBF algorithms lack heuristics to detect the inconsistencies that we have found in hard random unsatisfiable formulas. We believe that the heuristics we have developed could be incorporated into a complete QBF solver. It should be possible to generalize these heuristics for (a) more general random QBF formulas and, (b) some class of real-world formulas.

References

[1] Y. Boufkhad, O. Dubois, Y. Interian, and B. Selman. Regular random k-sat: Properties of balanced formulas. *Journal of Automated Reasoning*, 2005.

[2] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial*

intelligence/Innovative applications of artificial intelligence, pages 262–267. American Association for Artificial Intelligence, 1998.

[3] H. Chen and Y. Interian. A Model for Generating Random Quantified Boolean Formulas. In *Proc. of 9th International Joint Conference on Artificial Intelligence (IJCAI05)*, 2005.

[4] G. Denquen and O. Dubois. kcnfs: An efficient solver for random k-sat formulae. *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003,LCNS, Selected Revised Papers*, 6:486–501, 2003.

[5] R. Feldmann, B. Monien, and S. Schamberger. A distributed algorithm to evaluate quantified boolean formulae. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, (AAAI'00)*, pages 285–290, 2000.

[6] I. P. Gent, H. H. Hoos, A. G. D. Rowley, and K. Smyth. Using stochastic local search to solve quantified boolean formulae. In *CP*, pages 348–362, 2003.

[7] I. P. Gent and T. Walsh. Beyond NP: the QSAT phase transition. *Proceedings of AAAI 1999*, 1999.

[8] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artif. Intell.*, 145(1-2):99–120, 2003.

[9] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. *In Proc. 10-th National Conf. on Artificial Intelligence*, pages 459–465, 1992.

[10] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, Providence RI, 1993.

[11] L. Simon, D. Le Berre, M. Narizzano, and A. Tacchella. The second qbf solvers comparative evaluation. In *Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT2004)*, 2004.