Using Stochastic Local Search to Solve Quantified Boolean Formulae

Ian P Gent[†], Holger H Hoos[‡], Andrew G D Rowley[†], and Kevin Smyth[‡]

[†]University of St. Andrews
Fife, Scotland
{ipg, agdr}@dcs.st-and.ac.uk

[‡]University of British Columbia
Vancouver, Canada
{hoos, ksmyth}@cs.ubc.ca

Abstract. We present a novel approach to solving Quantified Boolean Formulae (QBFs), exploiting the power of stochastic local search methods for SAT. This makes the search process different in some interesting ways from conventional QBF solvers. First, the resulting solver is incomplete, as it can terminate without a definite result. Second, we can take advantage of the high level of optimisations in a conventional stochastic SAT algorithm. Our new solver, WalkQSAT, is structured as two components, one of which controls the QBF search while the other is a slightly adapted version of the classic SAT local search procedure WalkSAT. The WalkSAT component has no knowledge of QBF, and simply solves a sequence of SAT instances passed to it by the QBF component. We compare WalkQSAT with the state-of-the-art QBF solver QuBE-BJ. We show that WalkQSAT can outperform QuBE-BJ on some instances, and is able to solve two instances that QuBE-BJ could not. WalkQSAT often outperforms our own direct QBF solver, suggesting that with more efficient implementation it would be a very competitive solver. WalkQSAT is an inherently incomplete QBF solver, but still solves many unsatisfiable instances as well as satisfiable ones. We also study run-time distributions of WalkQSAT, and investigate the possibility of tuning WalkSAT's heuristics for use in QBFs.

1 Introduction

Stochastic local search (SLS) methods are an area of continuing interest in the satisfiability (SAT) community. While not guaranteed to return a solution (nor to determine unsatisfiability), they can often be more effective than complete methods, as they are not restricted by the need to cover the entire search space systematically. It is natural to wonder if SLS methods can be applied to Quantified Boolean Formulae (QBF) problems. QBF is a generalisation of SAT with applications in areas such as hardware verification, planning, and games. Variables in a QBF instance can be either existentially or universally quantified. Put simply, with details to follow below, a QBF problem is satisfiable if the existential variables can be set to satisfy the instance, in SAT terms, for all possible instantiations of the universal variables.

Unfortunately, the application of SLS methods to QBF is problematic. The most pressing problem is that individual search states are not simply assignments of variables to the two truth values. Instead, the most natural representation of a search state is as a strategy, defining the values of the existential variables for each possible instantiation of the universals. However this is an infeasibly large object except when there are a very small number of universal variables. Despite the difficulties in applying SLS techniques to QBF solving, there are compelling reasons for doing so. Search in a QBF is a search for many satisfying assignments for a variety of very closely related SAT instances. Not only can SLS methods often perform these searches very fast, they can naturally take advantage of solutions to previous instances as starting points for the current search.

Our incomplete QBF solver, called WalkQSAT, is structured as a collaboration between two components. The first component, the QBF engine, performs a backjumping search based on a successful method from the literature called Conflict and Solution Directed Backjumping (CSBJ) [1]. The second component, the SAT engine, is used as an auxiliary search procedure to find satisfying assignments quickly (details of these components, and how they interact are given later in the paper). We use WalkSAT [2] as the SAT engine for WalkQSAT, although a whole family of QBF algorithms can be designed by using other algorithms for the SAT engine. WalkQSAT has the following properties. If it returns True (T) or False (F) given an instance, that instance is guaranteed to be true or false respectively. If it returns Unknown (U), then the truth or falsity of the instance could not be verified within the given constraints. WalkQSAT is naturally more likely to successfully solve true instances, given that false instances are more likely to contain more states in which WalkSAT will not be able to find a solution, but some false instances can be solved nevertheless. As we will see, WalkQSAT can outperform state-of-the-art solvers on some instances.

This paper introduces for the first time the study of SLS methods for solving QBF instances. Even so, we are in some instances able to outperform the state-of-the-art solver QuBE-BJ. While typically our performance is not as good as QuBE-BJ, we have shown the potential of stochastic local search methods for QBF.

2 Background

A QBF is presented as a Boolean formula in conjunctive normal form (CNF) with a prefix of quantifiers. More formally, a QBF is of the form $\mathbf{Q} = QB$ where the prefix $Q = q_1x_1q_2x_2 \ldots q_nx_n$ is a sequence of pairs of quantifiers $q_i \in \{\forall, \exists\}$ and propositional variables x_i, and B is a propositional formula in CNF. A CNF formula is a conjunction of clauses; each clause is a disjunction of literals, and each literal is a propositional variable in negated or unnegated form. Within the prefix Q every variable in B is quantified exactly once by either an existential or universal quantifier. These variables are then known as existentials and universals respectively.

The satisfiability of the CNF part B of a QBF is defined just as in SAT, i.e. B is satisfied if every clause contains a true literal. However, the QBF is only satisfied if appropriate values can be given to the existentials to allow B to be satisfied for any instantiation of the universals. For this, the order of variables in the prefix is critical. We can define the truth of a QBF recursively. A QBF $\mathbf{Q}$ with an empty prefix is true iff its CNF part B is satisfied. If $\mathbf{Q}$ has a non empty prefix, there are two cases. A QBF $\exists x_1 Q_1$ is true iff either $Q_1[x_1 := T]$ or $Q_1[x_1 := F]$ is true; while a QBF $\forall x_1 Q_1$ is true iff both $Q_1[x_1 := T]$ and $Q_1[x_1 := F]$ are true. For example, a QBF $\forall x_1 \exists x_2 \forall x_3 \exists x_4 B$ is true if and only if for both True (T) and False (F) assignments to x_1, there is an assignment (T or F) to x_2, where for both assignments to x_3 there is an assignment to x_4 for which B is satisfied.

Solving a QBF can be seen as finding a winning strategy in a two player game between universal and existential quantifiers. The variables are the pieces and the assignments the moves. Existential wins if the assignments to the variables leaves a satisfied literal in every clause (i.e. B is satisfiable) and universal if the assignments leave a clause containing all negative literals. The order of the moves are dictated by the order in which the variables appear in the prefix of the QBF. The QBF is satisfiable if the existential player can find a strategy in which she can win no matter what moves the universal player makes; it is unsatisfiable if this is not the case.

2.1 Stochastic Local Search for SAT

Stochastic Local Search (SLS) algorithms for satisfiability (SAT) attempt to solve a given CNF formula B by iteratively changing, or flipping, the value assigned to variables in B such that the number of clauses that remain unsatisfied by the assignment is minimised. The selection of the variable to be flipped in each search step is typically performed using a randomised greedy mechanism. The WalkSAT family [2, 3] comprises some of the most widely studied and best-performing SLS algorithms for SAT; it is based on a randomised greedy local search procedure that flips a variable from an unsatisfied clause in each search step.

SLS-based solvers for SAT are typically incomplete, i.e., they cannot determine the unsatisfiability of a given formula, but may find a satisfying assignment, if it exists, rather efficiently. Thus, applied to an unsatisfiable formula, they will eventually terminate and return Unknown. For satisfiable formulae, True is returned (along with the respective assignment) if a satisfying assignment is found within the given resource limits and Unknown is returned otherwise. The latter particularly happens if the algorithm gets stuck in a local minimum of the underlying evaluation function.

2.2 Backtrack Search for QBFs

Backtrack search attempts to determine the truth of a QBF by assigning truth values to variables and simplifying the formula until it is vacuously true or vacuously false. Then, if false is found, all variables up to and including the last existential variable assigned are unassigned, and the last existential is assigned to the opposite value and the process is repeated. Similarly if true is found, all variables up to and including the last universal are unassigned and the last universal is assigned to the opposite value and the process is repeated. Once a variable has been assigned both True and False, the combination of the results of these two assignments is returned dependant on the quantification of the variable. These can be seen in Figure 1. The point at which the decision is made to assign a variable True or False is known as a branch point.

A QBF is vacuously true if it consists of an empty set of clauses. It is vacuously false if the set of clauses contains either a clause with no literals (empty clause) or a clause with only universal literals (all universal clause). An all universal clause cannot be satisfied since the clause must be true for all assignments to the universals and so

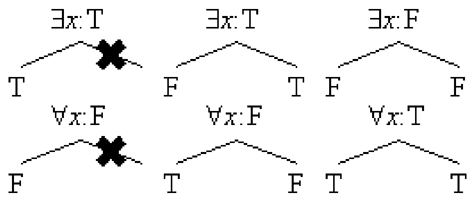

Fig. 1. The possible branching and return values for CSBJ. In the first column the second assignment is not tried since the first assignment yields enough information.

it will be unsatisfied if all the literals are assigned false. (The exception is tautologous clauses, but we remove these during preprocessing.)

If a variable is assigned True, any clause containing the literal of the variable with the positive sign can be removed, and the variable can be removed from any clause containing the literal of the variable with the negative sign. Additionally, the variable can be removed from its quantifier and empty quantifiers can be removed. In a backtrack search, variables are assigned in turn until the QBF is vacuously true or vacuously false. If true (respectively false), the variables are then unassigned until a universal (respectively existential) assignment is undone. This assignment is then reversed and the variables are again assigned until true or false is found and the process is repeated.

A unit clause is a clause that contains only one literal. This literal must be assigned true to eventually get a vacuously true state, otherwise the clause will be empty. A single existential clause is a clause that contains only one existential literal and in which all universal literals are quantified further right in the prefix than the existential. The existential literal must be assigned true because if it is assigned false the clause will become all universal and thus unsatisfied. A pure literal is found when every occurrence of a literal within the set of clauses has the same sign. An existential pure literal can be assigned true. If we reach the vacuously false state then we can be sure that we could have done no better in assigning the literal false, and so backtracking is unnecessary on the variable. A universal pure literal can be assigned false. If we reach the vacuously true state then we can be sure that we would have done no worse in assigning the pure literal true, and so again, no backtracking is required.

Conflict and Solution Directed Backjumping (CSBJ) for QBFs [1] reduces the number of backtracks performed. This is done by calculating either a conflict set or a solution set. A conflict set is a set of existential variables that caused the conflict, i.e. the empty or all universal clause. A solution set is a set of universal variables such that all clauses not satisfied by the current existential assignment are satisfied by at least one of the universal variables. On returning to an existential branch point (in the case of a conflict), or a universal branch point (in the case of a solution), a backtrack need only be performed if the variable assigned at the branch point is in the set. This technique has been shown to be useful in the solving of QBFs [4], in particular on random instances with three or more quantifiers, and on 'real world' instances. The cover set used in solution directed backjumping is not unique [5]. It is important that a small cover set is chosen, to reduce the number of universal backtrack points.

Finally, we mention the Trivial Truth method [6]. This is a method for using a SAT algorithm within a QBF solver. With this technique a counterpart $\mathbf{E}$ to the QBF $\mathbf{Q}$ is kept where $\mathbf{E}$ is the same as $\mathbf{Q}$, but with the universals removed. If a solution to $\mathbf{E}$ can be found, this is also a solution to $\mathbf{Q}$. however, if no solution to $\mathbf{E}$ can be found, nothing has been gained, since $\mathbf{Q}$ could still be true. If no solution is found, the results are discarded. This SAT search is potentially wasteful since the results of useful search are discarded if the result is false.

3 WalkQSAT

WalkQSAT is in essence an implementation of conflict- and solution-directed backjumping (CSBJ) in QBF. However, it uses an auxiliary stochastic SAT solver, WalkSAT, to guide its search. WalkSAT is used to solve the current reduced QBF instance viewed purely as a SAT problem, i.e. treating each universal variable as an existential (unlike trivial truth where the universal variables are removed). This solution is used to set the values of variables in the CSBJ search; when a variable is heuristically chosen in the QBF solver, the value assigned to the variable is the value assigned by WalkSAT in this returned solution. This has two consequences. First, it is possible for WalkSAT to fail to find a solution, either because there is none or because WalkSAT times out. This leads to the inherent incompleteness of WalkQSAT. The second consequence is more positive. Where WalkSAT finds a satisfying assignment, this guarantees that if the same assignments are given to the variables in the QBF search, the vacuously true state will be reached. Thus WalkSAT is being used for more than purely heuristic guidance. From a vacuously true QBF found this way, we continue as CSBJ normally would. That is, we backjump to the most recent universal variable in the solution set. After backjumping, WalkSAT is called to determine the next set of assignments to guarantee a vacuously true state.

It is straightforward to deal with the case that WalkSAT fails to solve an instance. When WalkSAT times out, it returns the value U for unknown. No further attempt is made to solve the corresponding node in the CSBJ search, but Figure 2 shows the additional possibilities and the associated return states allowing for Unknown values at branch points in the QBF search procedure. If an U is returned for the first assignment, WalkQSAT always tries the second assignment, sometimes being able to determine the result. If not, U is passed back.

It might seem that it is impossible for WalkQSAT ever to determine the falsity of a QBF instance, as WalkSAT cannot determine unsatisfiability of SAT instances. However, as any CSBJ algorithm does, WalkQSAT implements single-existential propagation, which can lead to a contradiction. From this a conflict set can be calculated for backjumping, and if the first variable assigned is ever backjumped over, the problem has been proved false. Thus, with QBFs, we encounter a different kind of incompleteness than that of WalkSAT. Specifically: if False is returned, the problem is definitely false; if True is returned, the problem is definitely true; but the solver can still return Unknown if no proof is found. In practice, we found that WalkQSAT was often able to determine falsity of QBF instances. For this, the use of conflict-directed backjumping

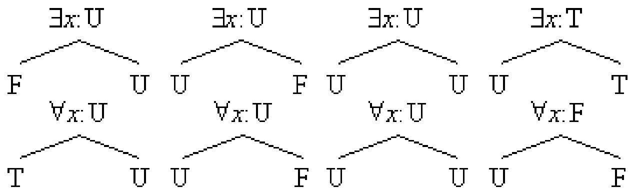

Fig. 2. The additional branching and return values for WalkQSAT. Note how the solver recovers in the last column.

is a help, as an unknown value returned at a node not in the conflict set does not prevent falsity being proved.

The key issue in design and implementation was to keep WalkSAT and WalkQSAT in step. At any node in its search tree, WalkQSAT needs to get a satisfying assignment to B that is consistent with the values of variables that have been set by branching or propagation at higher levels of the search tree. So we cannot let WalkSAT continue solving the original instance of the QBF viewed as a SAT problem. Instead, we notify WalkSAT each time a variable is assigned, via an interface call *Fix(literal)*. We implemented a variant of WalkSAT in which fixed variables could not be changed. After each variable is set either by branching or propagation, *Fix* is called. This means that whenever WalkQSAT calls WalkSAT, the SAT search is only on variables free at this node in the search tree. When WalkQSAT backtracks over a fixed variable, a corresponding *Release(literal)* call is made. This results in two data structures being maintained to keep the current search state, one of which is a data structure optimised for efficient complete search, while the other is optimised for efficient local search.

One aspect of WalkSAT we found to affect performance was the starting position of its second and later searches. Instead of a random position each time, we found it much more effective to start the search from the last assignment visited during the previous search, except for changes forced by *Fix* calls. This is natural, as often there will be relatively few *Fix* calls between calls to WalkSAT, so a solution at the last node is likely to be a near-solution at the next node. However, since in normal use WalkSAT (for sufficiently high noise settings) is insensitive to whether it is restarted or simply left to run [7, 8], it may be surprising that this use of starting position seemed to be necessary. However, this insensitivity to restarts does not hold for very short runs, particularly if these are started very close to a solution.

Figure 3 shows pseudocode for WalkQSAT. This is similar to the pseudocode for QuBE-BJ [1] since WalkQSAT is also a backjumping algorithm. A call to *ChooseLiteral* additionally passes the assignments returned by WalkSAT, and these are used to determine the sign of the literal to be used in the next assignment. The actual literal chosen can be independent of the results of WalkSAT. $InitWr$ calculates the conflict or solution set, given the false (F) or true (T) result respectively. If $InitWr$ is called with the unknown (U) result, it should return the empty set, since backtracking must be performed instead. The function $BackJump$ correctly deals with the case where unknown has been returned, as shown in Figure 2, by immediately backtracking on the last literal assigned, or if the literal has already been backtracked upon, passing back the unknown result. The variable $last$ is used to determine the last action performed by the algorithm. This way, WalkSAT is only called after backjumping has been performed. Other functions have the same meaning as was described in [1], repeated here for completeness:

- **Q** is a global variable storing the QBF in the current state, initially set to the input QBF.
- $Stack$ is a global variable storing the search states so far, initially empty.
- *T*, *F*, *U*, UNDEF, SINGLE∃, PURE, L-SPLIT, R-SPLIT, CHOOSE and BACK are constants.
- $Extend(l)$ removes $\overline{l}$ from all clauses in which it appears, removes all clauses containing l and pushes l and **Q** onto the stack.
- $Retract()$ gets l and **Q** from the top of the stack and undoes all work done by $Extend(l)$.

Note that WalkSAT is only one example of a solver that could be used: any SAT solver can be used as long as it implements the interface defined here, and that it never returns an incorrect value. While in this paper we restrict ourselves to the use of WalkSAT, it will be interesting to see if other SAT solvers can perform well in this framework.[1]

The method described here could easily be mistaken for trivial truth, in that a truth assignment is found by WalkSAT. This is not the case however, since trivial truth finds a truth assignment only involving the existentials. WalkSAT finds a truth assignments assuming the universals are existentials. It is this key difference that allows WalkQSAT to get more information from the SAT solver than trivial truth. If trivial truth finds a satisfying assignment, search can be cut off on the current branch. However, if trivial truth fails to find such an assignment, the results of the SAT search are disregarded and search continues. When WalkQSAT finds a satisfying assignment, the results are used to guide search, and are not just discarded. This is clearly different to trivial truth and less wasteful of resources. Another difference with trivial truth is that, if WalkSAT fails to find a satisfying assignment, we stop searching and backtrack, returning Unknown.

4 Experimental Methodology

We explored the performance of WalkQSAT, both in its own terms and against an existing state-of-the-art solver for QBF. To test WalkQSAT experimentally, we need both a good set of benchmark instances, and a good methodology which gives a fair understanding of WalkQSAT with respect to the state of the art. This is particularly important given that there are a number of parameters which can affect WalkQSAT's performance, and that as a randomised procedure it gives different performance on each run. To compare against the state of the art, we compare results with the complete solver QuBE-BJ [1], an implementation of CSBJ. We chose QuBE-BJ because, like WalkQSAT, it is

[1] This is why we deal correctly with the possibility of WalkSAT returning F in our pseudocode: while WalkSAT can never return F, other SAT solvers can.

```
function WalkQSAT(QBF Q̂)
    Q:= Q̂;
    Stack := Empty stack;
    last := CHOOSE;
    (res, assignments) := WalkSAT();
    if (res ≠ T) return res;
    do
        res := Simplify();
        if (last = BACK and res = UNDEF)
            (res, assignments) := WalkSAT();
            if (res = T) res := UNDEF;
        if (res = UNDEF)
            l := ChooseLiteral(assignments); last := CHOOSE;
        else
            l := Backjump(res); last := BACK;
        if (l ≠ UNDEF)
            Extend(l); Fix(l);
    while (l ≠ UNDEF);
    return res;

function BackJump(res)
    wr := InitWr(res)
    while (Stack is not empty)
        l := Retract(); Release(l);
        if (l ∈ wr or res = U)
            if (res = F and |l|.type = ∃) or
                (res = T and |l|.type = ∀) or
                (res = U)
                if (|l|.mode = SINGLE∃) or (|l|.mode = R-SPLIT)
                    wr := (wr ∪ |l|.reason)/{l, l̄}
                    if (|l|.result = U) res := |l|.result;
                if (|l|.mode = L-SPLIT)
                    |l|.result = res;
                    |l|.mode = R-SPLIT;
                    |l|.reason := wr;
                    return l;
            else wr := wr/l;
    return UNDEF

function Simplify()
    do
        Q' = Q;
        if (Q is vacuously false) return F
        if (Q is vacuously true) return T
        if (Q contains a single existential literal l)
            |l|.mode := SINGLE∃; Extend(l); Fix(l);
        if (Q contains a pure existential literal l)
            |l|.mode := PURE; Extend(l); Fix(l);
        if (Q contains a pure universal literal l)
            |l|.mode := PURE; Extend(l); Fix(l);
    while (Q' ≠ Q)
    return UNDEF
```

Fig. 3. The WalkQSAT algorithm. This is similar to the pseudocode for QuBE-BJ [1], but with modifications made to account for SAT solver calls, and dealing with Unknown (U) return values. **Q** and $Stack$ are global variables.

a backjumping algorithm and so makes for a good comparison. We do not know of a solver which is known to be better than QuBE-BJ, so our comparison is with the state of the art. We undertook benchmark tests on both random and structured problems. The latter came from QBFLib (http://www.mrg.dist.unige.it/QBFLIB/), and we used all instances except the robot problems. For randomised instances, we used Gent and Walsh's Model A [9], because it has been commonly used in previous literature and is well understood. The parameters used were 20 variables per quantifier, 4 quantifier alternations with the universal outermost, 5 variables per clause and a number of clauses from 25 to 450 in steps of 25.

WalkQSAT has a number of parameters that must be set for each run. These affect how the search performed by WalkSAT is carried out. In particular, there is the MaxFlips parameter, which is the number of flips WalkSAT will perform before returning U, and the noise parameter p which affects the level of randomisation vs. hill-climbing. In SAT, MaxFlips is not an important parameter because when using close-to-optimal noise settings, very large MaxFlips settings generally work well [8]. The setting of MaxFlips affects performance in QBF because there is a tradeoff between allowing WalkSAT enough time to solve each instance and spending too much time in wasted searches. Many of the tested subinstances will be unsatisfiable, and extra flips are entirely wasted. The noise parameter is often critical for applications of WalkSAT, and the default value of 0.5 sometimes gives very poor performance. To set these parameters to poor values could give an unduly bad impression of how WalkQSAT performs. On the other hand, to optimise performance on all instances would give an unduly good impression: in practice we cannot optimise parameters when presented with an instance that needs to be solved just once.

To resolve this dilemma, we follow a practice suggested by Hoos [7], of performing a coarse optimisation on a small subset of the instances. To this end, we varied MaxFlips from $1 \times n$ to $50 \times n$, where n is the number of variables in the instance, and noise from 0 to 0.75 on a random instance and an individual structured instance. In neither case did performance seem particularly sensitive to the settings of these parameters. We observed that settings of $10 \times n$ and 0.5 gave good performance in both cases, and we use these values in all experiments we report for WalkQSAT in this paper.

Since WalkQSAT is a randomised procedure, through its use of WalkSAT, data from an individual run could be misleading. Instead we are interested in the entire distribution of data from a number of runs. Throughout this paper we report results on 100 runs of WalkQSAT on each instance we test. The only exceptions are those that QuBE-BJ failed to solve in 20 minutes on a 1GHz PC: for these we tested WalkQSAT in only 25 tries of 20 minutes.

The purpose of this work is to introduce a method for implementing an incomplete solver for QSAT, which has never been done before. We have therefore not investigated different variants of WalkSAT and it is unknown how these will work for QBF solving. It is hypothesised that the variant will not be too important, since the work done by WalkSAT after the first run appears minimal. For all the experiments, we used the Novelty^{+} variant of WalkSAT [7]. This variant has been shown to perform very well on many SAT problems, and it is left as further work to examine the performance of other variants in WalkQSAT [7, 8]. All experiments were performed on a cluster of 1

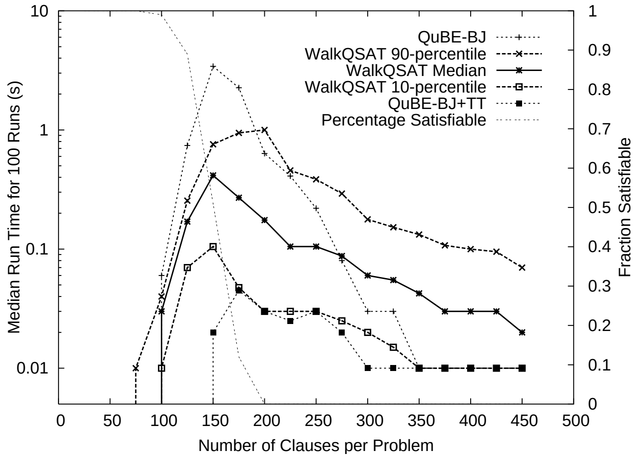

Fig. 4. Phase transition for various algorithms on random problems from Model A (n=20 per quantifier,h=5,outermost quantifier universal,4 quantifier alternations,l=25..450 in steps of 25.

Ghz computers with 512MB RAM running Linux kernel 2.4.7-10 and GNU gcc version 2.96. We used QuBE-BJ version 1.0 (http://www.mrg.dist.unige.it/˜qube/Download/download.html). The timeout was 20 minutes (1200 seconds) for all runs.

5 Experimental Results

WalkQSAT was compared to QuBE-BJ and QuBE-BJ with trivial truth. Figure 4 shows the phase transition commonly observed for random problems. WalkQSAT performs well here compared to QuBE-BJ but does not often outperform QuBE-BJ with trivial truth, although it often gives the same performance at the 10-percentile range. We conjecture that where WalkQSAT achieves the same performance as QuBE-BJ with trivial truth, it finds the same assignments that trivial truth makes, and so solves the QBF without the use of universals.

This last point can be seen even more clearly in Figure 5 where the 10-percentile errorbar extends to the same run time as QuBE-BJ with trivial truth on some instances. This figure also shows that WalkQSAT can outperform QuBE-BJ without trivial truth, even on false instances, which is a rather surprising result.

Figure 6 shows performance results of WalkQSAT on structured instances. Here, WalkQSAT only performs better than QuBE-BJ near the 10-percentile and only on a few problems. There are some sets of structured instances on which WalkQSAT shows very little variation in run time; this can be seen in the diagonal lines of data points for WalkQSAT for which errorbars are not visible, with corresponding crosses for QuBE-BJ with trivial truth. (these correspond to the CHAIN instances). WalkQSAT can perform better than QuBE-BJ with trivial truth, which illustrates the differences between WalkQSAT and backjumping with trivial truth. WalkQSAT outperforms QuBE-BJ with

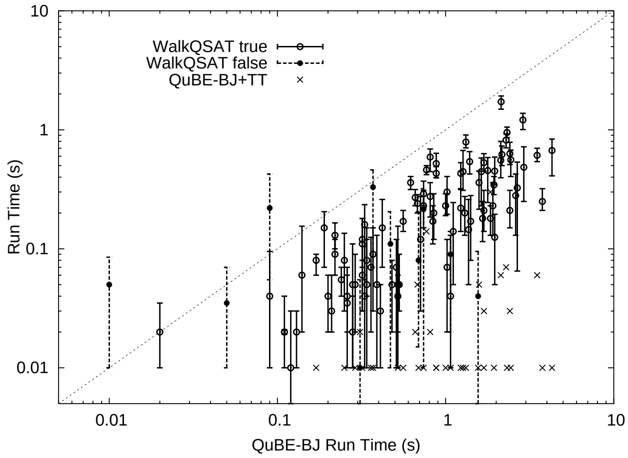

Fig. 5. Run-times for QuBE-BJ vs. WalkQSAT and QuBE-BJ+TT on random model A instances with 125 clauses. The error bars for the WalkQSAT run times indicate the range between the 10 and 90 percentiles of the underlying run-time distributions obtained for the respective instances. The diagonal line indicates equal run-time for QuBE-BJ and the other algorithm; points below (above) this line represent instances on which WalkQSAT or QuBE-BJ+TT, respectively, are faster (slower) than QuBE-BJ.

trivial truth on 168 runs on 13 different instances (i.e. 168 runs out of 1300 runs of WalkQSAT in total), and QuBE-BJ on 340 runs on 20 different instances, out of the 257 structured instances we tested.

As was said previously, where QuBE-BJ and QuBE-BJ with trivial truth did not solve a structured instance before the timeout, WalkQSAT was run on this instance to see if it could solve it. On one problem, TOILET10.1.iv.20 (true), WalkQSAT was able to solve it in 27.37 seconds only once out of 100 runs. On the other 99 runs, WalkQSAT timed-out. On one other problem, szymanski-16-s, QuBE-BJ both with and without trivial truth could not solve the problem due to insufficient memory. On this problem WalkQSAT solved the instance 12 out of 100 times with a median run time of 16.595 seconds.

In order to understand the run-time behaviour of WalkQSAT in more detail, we studied run-time distributions (RTDs) for individual problem instances following the methodology by Hoos and Stützle [8]. Since WalkQSAT, like WalkSAT, is a stochastic algorithm, when applied to the same instance, its run-time will vary stochastically. It is known that for WalkSAT, if sufficiently high noise parameter settings are used, the RTDs are well approximated by exponential distributions [7]. As can be seen in Figures 7 and 8, this is not the case for WalkQSAT. Although considerable variability in run-time can typically be observed, the right tail of the RTDs tends to be much skinnier, indicating that the probability of very long runs (compared to the average or median run-time) is very small. Interestingly, this appears to hold for satisfiable and unsatisfiable, random and structured instances. It implies that, different from several state-of-the art

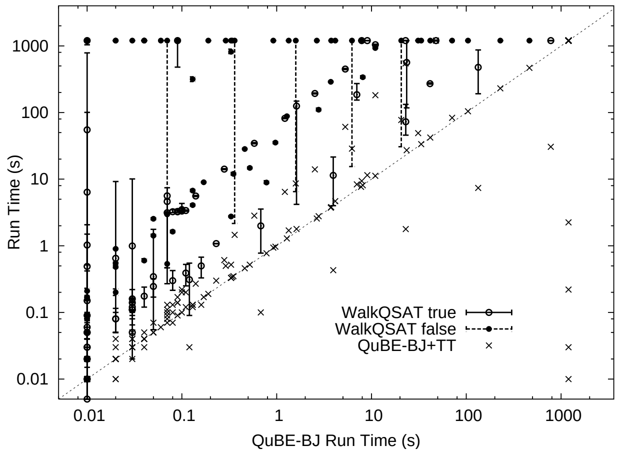

Fig. 6. Performance of WalkQSAT vs. QuBE-BJ and QuBE-BJ+TT on structured QBFLIB instances. See Figure 5 for more details. Where a circle appears without error bars, there is little or no variation in run time over 100 runs.

randomised systematic search algorithms for SAT, simple restarting strategies will not improve the performance of the algorithm.

We also note that when measuring only the total number of WalkSAT steps, we obtain run-length distributions that have substantially higher variability. But as can be seen from our RTD results, this large amount of variability present in the WalkSAT runs is reduced, rather than amplified when WalkSAT is used within the CSBJ framework.

6 A WalkSAT heuristic for QBF

The evaluation function of WalkSAT is usually the number of unsatisfied clauses, u. To help reduce the number of universal variables assigned, and so help solution directed backjumping, we alter the evaluation function of WalkSAT to be $\alpha u + \beta e$, where e is the number of satisfied clauses not satisfied by an existential. This gives us two parameters to tune, α and β. Since the important factor is α/β, α is set at 10, and β is varied.

It is found that on Rintanen's impl set of problems from QBFLIB, a value of $\beta > 0$ provides some significant improvements in run time. For example, on impl14, the median run time was 18.1 seconds with $\beta = 0$ and 2.07 seconds with $\beta = 1$. With increasing values of β, the median run time does not vary greatly, e.g. with $\beta = 1000$, the median run time is 2.13 seconds.

Further analysis shows that QuBE-BJ with trivial truth is more effective on these instances than without trivial truth. This is observed in Figure 6 as a set of points that appear below the diagonal representing the impl problems. The reasoning for the effectiveness of β on these problems is therefore likely to be that $\beta > 0$ makes WalkSAT behave more like trivial truth. Whilst this is the case, it has also been observed that $\beta > 0$ on other problems has no detrimental effect; this suggests that it is safe to use a

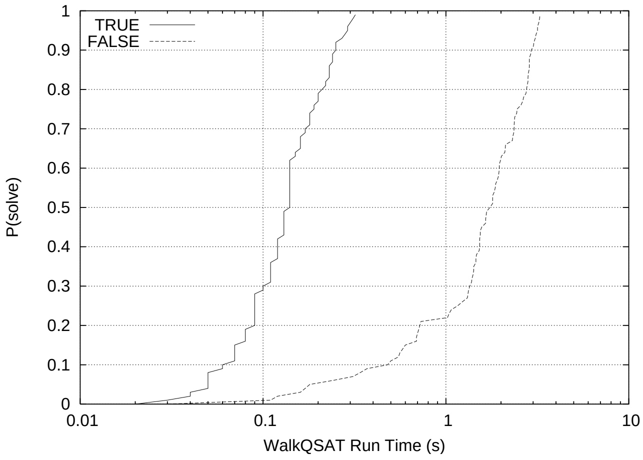

Fig. 7. WalkQSAT run-time distributions for typical satisfiable and unsatisfiable random instances from the phase transition.

high value of β just in case trivial truth helps on the problem. This is done with the risk that with increasing values of β, the time spent by WalkSAT in optimising the solutions may be better spent elsewhere and may result in WalkSAT not finding a solution at all. Of course, we still do not throw away valuable search as is done in the trivial truth method, but more unknown results may be returned.

7 Related Work

There has been an increase in interest in solving QBFs in recent years, starting with the introduction of a backtracking algorithm [6], followed by its application to planning problems translated to QBF [10]. The next big step was the introduction of backjumping for QBF [1], in particular solution directed backjumping which led to significant improvements in runtime. This gave rise to the next logical step of learning in QBF solvers [5, 11, 12], which provided improvements on some problems, whilst making others worse.

The WalkSAT algorithm family [2, 3, 8] comprises some of the most widely studied and best-performing SLS algorithms for SAT. Novelty^{+}, the WalkSAT variant used in WalkQSAT, was proposed in [7] and is based on the Novelty algorithm from [3].

Some interest in QBF has been on translation of QBF into SAT [13, 14]. This is naturally exponential in space, but the resultant SAT problem can be given to any SAT solver, including WalkSAT and other SLS solvers. To our best knowledge, WalkQSAT is the first QBF solver using SLS, except in this trivial sense.

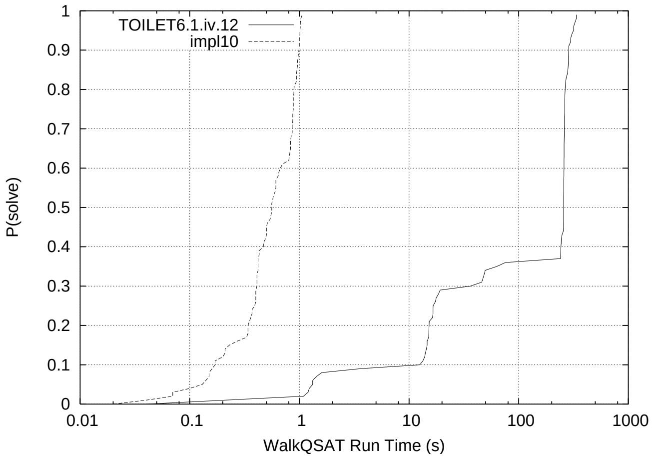

Fig. 8. WalkQSAT run-time distributions for two structured instances.

8 Conclusions and Future Work

In this paper, we have shown the potential of using stochastic local search methods in QBFs. We introduced WalkQSAT, a new QBF solver that combines Conflict and Solution Directed Backjumping (CSBJ) with a Stochastic Local Search procedure. We presented empirical evidence indicating that WalkQSAT, although an inherently incomplete algorithm, in most cases is able to correctly determine the satisfiability of a given QBF, and in many cases correctly determines unsatisfiability. (Like incomplete SLS algorithms for SAT and other problems, by design, WalkQSAT never gives an incorrect result, but may return "Unknown".) Although our implementation of WalkQSAT is not optimised for efficiency, it can solve several of the tested benchmark instances faster than QuBE-BJ, a state-of-the-art QBF solver based on CSBJ, and can even solve two instances that QuBE-BJ cannot. Our implementation of WalkQSAT is based on a CSBJ library that is known to be less efficient than QuBE-BJ. Improving this library should help to provide even better results.

A key issue in solution-directed backjumping is obtaining solutions which use only a small number of universal variables, minimizing the size of the solution sets. We have started to facilitate this by providing a modification of the evaluation function of WalkSAT, which improves WalkQSAT's performance on some instances. Furthermore, when WalkQSAT finds a solution quickly, it may be worth continuing the search to see if a better solution can be found, involving fewer universals. This could be seen as a solution-directed version of Schiex's 'stubbornness' for conflict-directed backjumping [15].

Acknowledgements

This work was partly supported by EPSRC grant GR/55382/01 and NSERC Individual Research Grant #238788. The first and third authors are members of the APES research group and thank other members. We thank the anonymous reviewers for their comments.

References

1. Guinchiglia, E., Narizzano, M., Tacchella, A.: Backjumping for quantified Boolean logic satisfiability. In: IJCAI-01. (2001) 275–281
2. Selman, B., Kautz, H.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: IJCAI-93. (1993) 290–295
3. McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: AAAI'97. (1997) 321–326
4. Guinchiglia, E., Narizzano, M., Tacchella, A.: An analysis of backjumping and trivial truth in quantified Boolean formulas satisfiability. In: AI*IA-01, Springer (2001) 111–122
5. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In: CP-2002, Springer (2002) 200–215
6. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified Boolean formulae. In: AAAI-98. (1998) 262–267
7. Hoos, H.: Stochastic Local Search - Methods, Models, Applications. PhD thesis (1998)
8. Hoos, H., Stützle, T.: Local search algorithms for SAT: An empirical evaluation. J. of Automated Reasoning **24** (2000) 421–481
9. Gent, I., Walsh, T.: Beyond NP: the QSAT phase transition. In: AAAI-99. (1999) 648–653
10. Rintanen, J.: Improvements to the evaluation of quantified boolean formulae. In: IJCAI-99. (1999) 1192–1197
11. Guinchiglia, E., Narizzano, M., Tacchella, A.: Learning for quantified boolean logic satisfiability. In: AAAI-2002. (2002) 649–654
12. Letz, R.: Lemma and model caching in decision procedures for quantified boolean formulas. In: TABLEAUX 2002. (2002) 160–175
13. Plaisted, D., Biere, A., Zhu, Y.: A satisfiability procedure for quantified boolean formulae. Discrete Applied Mathematics (2003) To appear.
14. Rintanen, J.: Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In: LPAR-01, Springer (2001) 362–376
15. Schiex, T., Verfaillie, G.: Stubbornness: a possible enhancement for backjumping and nogood recording. In: ECAI-94. (1994) 165–169