Using Stochastic Local Search to Solve Quantified Boolean Formulae

Ian P Gent[†] and **Holger H Hoos**[‡] and **Andrew G D Rowley**[†] and **Kevin Smyth**[‡]
[†]University of St. Andrews [‡]University of British Columbia
Fife, Scotland Vancouver, Canada
{ipg, agdr}@dcs.st-and.ac.uk {hoos, ksmyth}@cs.ubc.ca

Abstract

We present WalkQSAT, a solver for Quantified Boolean Formulae (QBFs) which makes use of stochastic local search. It 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. This means that any standard SAT solver, including complete methods, could also be used after minor adaptations. We show that WalkQSAT can outperform QuBE-BJ on some instances, and 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 distributions, and investigate the possibility of tuning WalkSAT's heuristics for use in QBFs.

1 Introduction

Stochastic local search 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 stochastic 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 stochastic local search 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 value of the universals. However this is an infeasibly large object except where there are a very small number of universal variables. There are serious reasons why we should want to exploit the power of local search techniques in a QBF solver. Search in a QBF is a search for very many satisfying assignments for a variety of very closely related SAT instances. Not only can stochastic local search often perform these searches very fast, they can naturally take advantage of solutions to previous instances as starting points for the current search.

Our approach here is to use WalkSAT [Selman and Kautz, 1993] as an adjunct to a special purpose QBF solver. Our incomplete solver, called WalkQSAT, is structured as a collaboration between two components. The QBF engine performs a backjumping search, while WalkSAT is used as an auxiliary search procedure to find satisfying assignments quickly. WalkQSAT has the following properties. If it returns True (T) or False (F) given an instance, that instance is guarenteed 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 return successfully on true instances, given that false instances are more likely to contain more false leaf nodes, but some false instances can be solved nevertheless. As we will see, WalkQSAT can outperform state-of-the-art solvers on some instances.

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 $\mathbf{Q} = QF$ where the prefix $Q = q_1 x_1 q_2 x_2 \ldots q_n x_n$ is a sequence of pairs of quantifiers $q_i \in \{\forall, \exists\}$ and propositional variables x_i, and F 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 F is quantified exactly once by either an existential or universal quantifier. These variables are then known as existentials and universals respectively.

The satisfaction of the CNF part F of a QBF is defined just as in SAT, i.e. if every clause contains a true literal. However, the QBF is only satisfied if appropriate values can be given to the existentials to allow F to be satisfied for any values 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 F 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.

2.1 Stochastic Local Search for SAT

Stochastic Local Search (SLS) algorithms for satisfiability (SAT) attempt to solve a given CNF formula F by iteratively changing, or flipping, the value assigned to variables in F such that the number of clauses that remain unsatisfied by the assignment is minimised. The selection of the variale to be flipped in each search step is typically performed using a randomised greedy mechanism. The WalkSAT family [Selman and Kautz, 1993; McAllester *et al.*, 1997] comprises some of the 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 asssignment is found within the given resource limits and Unknown is returned otherwise.

2.2 Backtrack Search for QBFs

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 it will be empty if all the literals are assigned false.

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. The assignment is then reversed and the variables are again assigned until a true or false leaf is found and the process is repeated.

A unit clause is a clause that contains only one literal. This literal must be assigned true for a path to a solution node, 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 occurence of a literal within the set of clauses has the same sign. An existential pure literal can be assigned true. If

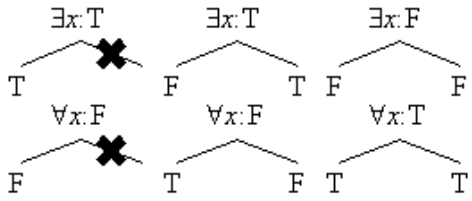

Figure 1: The possible branching and return values for CSBJ. In the first column the second branch is not tried since the first branch yields enough information.

we find a false leaf node 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 find a true leaf node 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 [Guinchiglia *et al.*, 2001b] 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 a 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 [Guinchiglia *et al.*, 2001a], 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 [Zhang and Malik, 2002]. 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 [Cadoli *et al.*, 1998]. 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 after a variable assignment is made, this can be used as a solution for this variable assignment. If no solution is found, the results are discarded.

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. pretending each universal variable to be existential. This solution is used to set the values of variables in the CSBJ search. 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 the next leaf node in the search tree will be satisfied, without backtracking. Thus WalkSAT is being used for

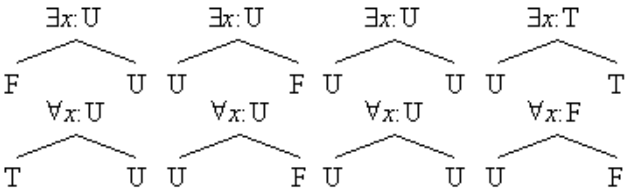

Figure 2: The additional branching and return values for WalkQSAT. Note how the solver recovers in the last column.

```
WalkQSAT(Q,MaxFlips,p)
   Q = Propagate(Q)
   If Q vacuously true Return (SolutionSet(Q),T)
   ElseIf Q vacuously false Return (ConflictSet(Q),F)
   a := array of current assignments
   w := WalkSAT(Q,MaxFlips,p,a)
   If w = T Return QSearch(Q, MaxFlips,p,a)
   ElseIf w = F Return (ConflictSet(Q),F)
   Else Return (∅,U)

QSearch(Q, MaxFlips,p,a)
   Q := Propagate(Q)
   If Q is vacuously true Return (SolutionSet(Q),T)
   ElseIf Q is vacuously false Return (ConflictSet(Q),F)
   v := variable in outermost quantifier
   Q := Q[v := a[v]]; and call Fix(v, a[v])
   (s1, r1) := QSearch(Q, MaxFlips,p,a)
   Q := UnPropagate(Q, v)
   Unassign v in Q and call Release(v)
   If (v ∈ ∃ and r1 = T) or (v ∈ ∀ and r1 = F)
      or (v ∉ s1 and r1 ≠ U)
      Return r1
   Else
      Q := Q[v := ā[v]]; and call Fix(v, ā[v])
      (s2, r2) := WalkQSAT(Q,MaxFlips,p)
      Q := UnPropagate(Q, v)
      Unassign v in Q and call Release(v)
      If (v ∈ ∃ and r2 = T) or (v ∈ ∀ and r2 = F)
      or (v ∉ s2 and r2 ≠ U)
         Return r2
      Else
         If (r1 = T and r2 = T)
            or (r1 = F and r2 = F)
            Return (s1 ∪ s2 − {v}, r1)
         Else Return (∅,U)
```

Figure 3: The WalkQSAT algorithm and its QSearch subroutine. True is represented as T, False as F and Unknown as U. $\exists$ and $\forall$ are respectively the sets of all existential and universal variables in **Q**. MaxFlips and p are standard WalkSAT parameters, p being the noise, while a is a boolean array for the return of the satisfying assignment found by WalkSAT. The call to WalkSAT returns T, F or U. The function Propagate assigns any unit and single existential clauses and pure literals appropriately, and calls *Fix* for each assignment made. Unpropagate reverses these assignments and calls *Release* for each assignment reversed. SolutionSet and ConflictSet calculate the appropriate sets.

more than purely heuristic guidance. From a satisfied leaf node found this way, we continue as CSBJ normally would. That is, we backjump to the most recent universal variable in the solution set. After taking the right branch, WalkSAT is called to determine the next path to take to a true leaf.

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 nodes in the tree. If an U is returned on the first branch, WalkQSAT always tries the second branch, sometimes being able to determine the result. If not, U is passed back up the tree.

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 root of the tree is ever backjumped to, the problem has been proved false. Thus, with QBFs, we encounter a different kind of incompleteness to 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 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 for the current settings of variables. So we cannot let WalkSAT continue solving the original instance of the QBF viewed as a SAT problem. Instead, we tell WalkSAT each time a variable is set, via an interface call *Fix(variable, value)*. 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 on 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(variable)* 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.

Figure 3 shows pseudocode for WalkQSAT. Note that WalkSAT is only an 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]

4 Experimental Methodology

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. To compare against the state of the art, we compare results with the complete solver QuBE-BJ [Guinchiglia *et al.*, 2001b]. We chose this because, like WalkQSAT, it is a backjumping algorithm and so makes for a good comparison. 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 as many instances as time permitted, using a selection of both true and false instances.

For randomised instances, we used Gent and Walsh's Model A [Gent and Walsh, 1999], because it is used many times 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. 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 be critical for application 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 [Hoos, 1998], 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 saw 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

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

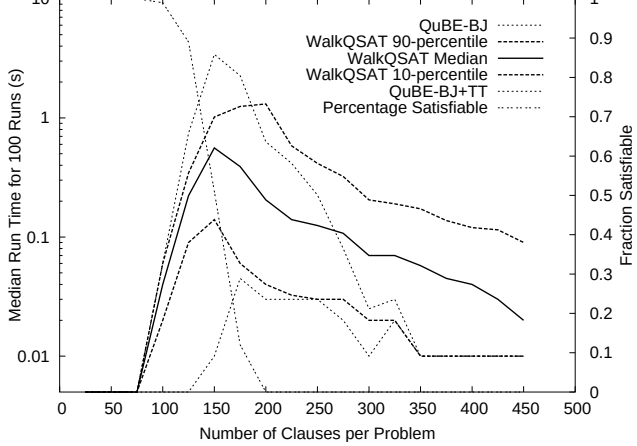

Figure 4: Phase transition for various algorithms on random problems from Model A.

results on 100 runs of WalkQSAT on each instance we test. The only exceptions are those that QuBE-BJ failed to solve in 1 hour cpu on a 1GHz PC: for these we tested WalkQSAT in only 25 tries of 1 hour each, which in all cases failed and for which we report no further results.

For all the experiments, we used the Novelty$^+$ variant of WalkSAT [Hoos, 1998]. This variant has been shown to perform very well on many SAT problems, but it is left as further work to examine the performance of other variants in WalkQSAT.

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 often gives the same performance at the 10-percentile range. It is conjectured that where WalkQSAT performs the same as QuBE-BJ with trivial truth WalkQSAT 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 figures 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 how WalkQSAT can outperform QuBE-BJ without trivial truth, even on false instances, which we find surprising.

The performance of WalkQSAT on structured instances is demonstrated by figure 6. Here, WalkQSAT only performs better than QuBE-BJ near the 10-percentile and only on a few problems. There are some problems on which there is very litte variation in the run time of WalkQSAT for all the instances in some sets of structured problems, as shown by diagonal lines of points where the errorbars are not visible (these are the CHAIN instances). WalkQSAT does better when compared to QuBE-BJ with trivial truth, which shows that WalkQSAT is not the same as backjumping with trivial truth. WalkQSAT outperforms QuBE-BJ on 63 runs on 12 different instances, and QuBE-BJ with trivial truth on 70 on 11 different instances, out of the 65 instances we tested.

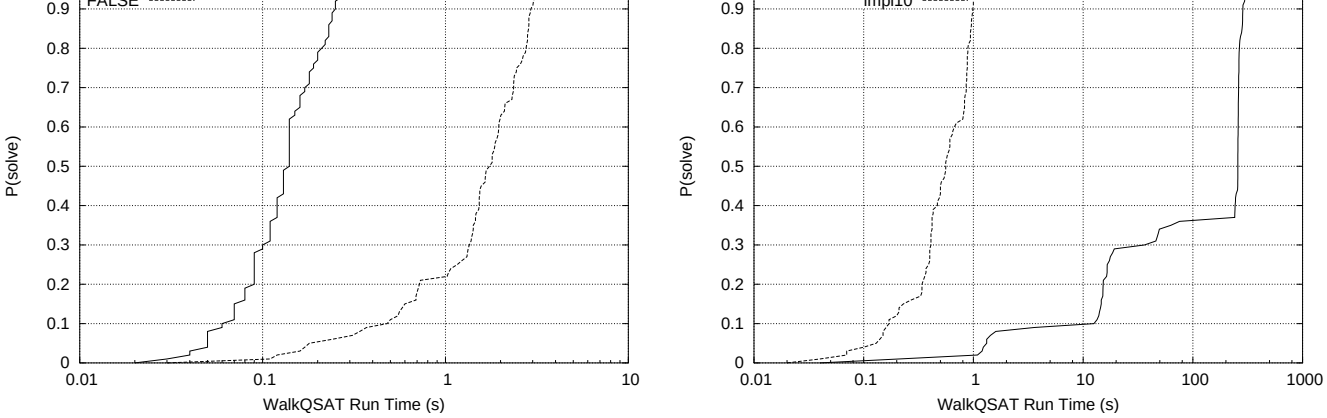

Figure 7: Left: WalkQSAT RTDs for typical satisfiable and unsatisfiable random instances from the phase transition (150 clauses). Right: WalkQSAT RTDs for two structured instances.

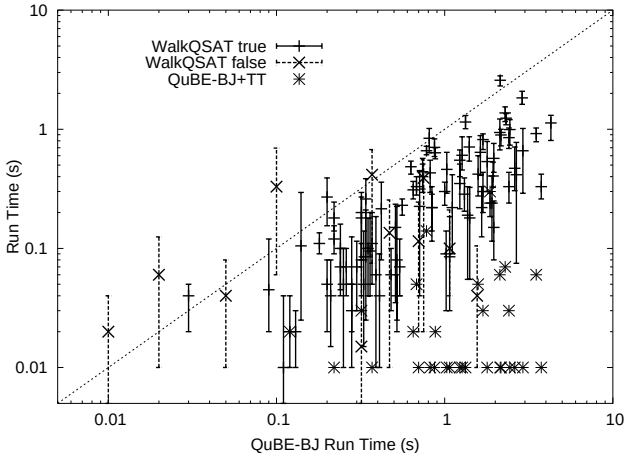

Figure 5: WalkQSAT vs. QuBE-BJ on random model A instances with 125 clauses. Error bars are shown on the WalkQSAT run times - these show the range between the 10 and 90 percentile run times of WalkQSAT.

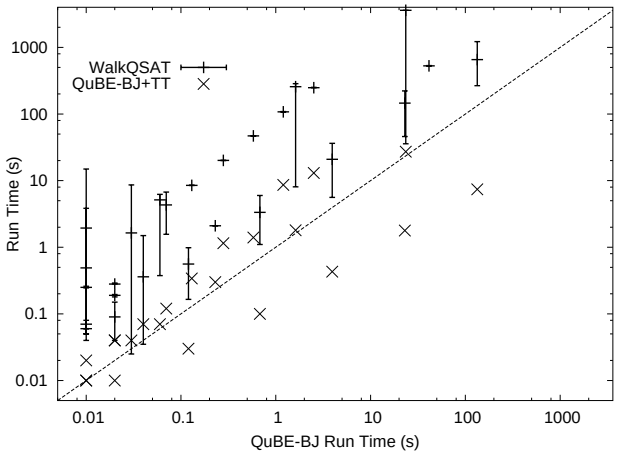

Figure 6: WalkQSAT vs. QuBE-BJ on structured QBFLIB instances. Only true instances are shown here – only 6 false instances were solved within 1 hour.

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 [Hoos and Stützle, 2000]. 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 [Hoos, 1998]. As can be seen in figure 7, 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 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 objective 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 objective 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 beta, the median run time does not vary greatly e.g. with $\beta = 1000$, the median run

time is 2.13 seconds.

Futher 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 effect; this suggests that it is safe to use a high value of β just in case trivial truth helps on these 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.

7 Related Work

There has been an increase in interest in solving QBFs in recent years, starting with the introduction of a backtracking algorithm [Cadoli *et al.*, 1998], followed by its application to planning problems translated to QBF [Rintanen, 1999]. The next big step was the introduction of backjumping for QBF [Guinchiglia *et al.*, 2001b], in particular solution directed backjumping, which lead to significant improvements in runtime. This gave rise to the next logical step of learning in QBF solvers [Guinchiglia *et al.*, 2002; Zhang and Malik, 2002; Letz, 2002], which provided improvements on some problems, whilst making others worse.

The WalkSAT algorithm family [Selman and Kautz, 1993; McAllester *et al.*, 1997; Hoos and Stützle, 2000] comprises some of the most widely studied and best-performing SLS algorithms for SAT. WalkSAT/SKC, the WalkSAT variant used in WalkQSAT, was proposed in [Selman and Kautz, 1993].

Some interest in QBF has been on translation of QBF into SAT [Plaisted *et al.*, 2003; Rintanen, 2001]. 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.

8 Conclusions and Future Work

In this paper, 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 some 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, a state-of-the-art QBF solver based on CSBJ. Our local implementation of CSBJ is based on the same library as WalkQSAT and is known to be less efficient than QuBE. 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 help in this by providing a modification of the objective function of WalkSAT, which helps on some instances. However, when WalkQSAT finds a solution quickly, it may be worth continuing 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 [Schiex and Verfaillie, 1994].

References

[Cadoli *et al.*, 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *AAAI-98*, pages 262–267, 1998.

[Gent and Walsh, 1999] I. Gent and T. Walsh. Beyond NP: the QSAT phase transition. In *AAAI-99*, pages 648–653, 1999.

[Guinchiglia *et al.*, 2001a] E. Guinchiglia, M. Narizzano, and A. Tacchella. An analysis of backjumping and trivial truth in quantified Boolean formulas satisfiability. In *AI*IA-01*, pages 111–122. Springer, 2001.

[Guinchiglia *et al.*, 2001b] E. Guinchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified Boolean logic satisfiability. In *IJCAI-01*, pages 275–281, 2001.

[Guinchiglia *et al.*, 2002] E. Guinchiglia, M. Narizzano, and A. Tacchella. Learning for quantified boolean logic satisfiability. In *AAAI-2002*, pages 649–654, 2002.

[Hoos and Stützle, 2000] H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *J. of Automated Reasoning*, 24(4):421–481, 2000.

[Hoos, 1998] H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, TU Darmstadt, 1998.

[Letz, 2002] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *TABLEAUX 2002*, pages 160–175, 2002.

[McAllester *et al.*, 1997] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *AAAI'97*, pages 321–326, 1997.

[Plaisted *et al.*, 2003] D. Plaisted, A. Biere, and Y. Zhu. A satisfiability procedure for quantified boolean formulae. *Discrete Applied Mathematics*, 2003. To appear.

[Rintanen, 1999] J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *IJCAI-99*, pages 1192–1197, 1999.

[Rintanen, 2001] J. Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In *LPAR-01*, pages 362–376. Springer, 2001.

[Schiex and Verfaillie, 1994] T. Schiex and G. Verfaillie. Stubbornness: a possible enhancement for backjumping and no-good recording. In *ECAI-94*, pages 165–169, 1994.

[Selman and Kautz, 1993] B. Selman and H. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *IJCAI-93*, pages 290–295, 1993.

[Zhang and Malik, 2002] L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *CP-2002*, pages 200–215. Springer, 2002.