An analysis of backjumping and trivial truth in quantified boolean formulas satisfiability*

E. Giunchiglia[1], M. Narizzano[1], and A. Tacchella[2]

[1] DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
[2] CS Dept., Rice University, 6100 Main St. MS 132, 77005-1892 Houston, Texas

Abstract. Trivial truth and backjumping are two optimization techniques that have been proposed for deciding quantified boolean formulas (QBFs) satisfiability. Both these techniques can greatly improve the overall performance of a QBF solver, but they are the expression of opposite philosophies. On one hand, trivial truth is a "look-ahead" policy: it is applied when descending the search tree to (try to) prune it. On the other hand, backjumping is a "look-back" policy: it is applied when backtracking to (try to) avoid useless explorations. Neither of these optimizations subsumes the other: it is easy to come up with examples in which trivial truth behaves much better than backjumping, and the other way around. In this paper we experimentally evaluate these two optimizations both on randomly generated and on real world test cases.

1 Introduction

Trivial truth [1, 2] and backjumping [3] are two optimization techniques that have been proposed for deciding quantified boolean formulas (QBFs) satisfiability. Both these techniques can greatly improve the overall performance of a QBF solver, but they are the expression of two opposite philosophies. On one hand, trivial truth is a "look-ahead" policy: it is applied when descending the search tree to (try to) prune it. On the other hand, backjumping is a "look-back" policy: it is applied when backtracking to (try to) avoid useless explorations. Neither of these two optimizations subsumes the other: it is easy to come up with examples in which trivial truth behaves much better than backjumping, and the other way around. In this paper we experimentally compare trivial truth and backjumping. The comparison is done using the QUBE system [4]. While trivial truth is implemented by various QBF solvers like EVALUATE [1] and QSOLVE [5], QUBE is –as far as we know– the only system with a backjumping optimization built in. The experimental analysis is performed on a combination of random kQBFs (see [2]), generated according to "model A" proposed by Gent and Walsh in [6], and on the real world instances proposed by Rintanen in [7]. The conclusions are that

- on random kQBFs, backjumping becomes more effective as the value of k increases, while trivial truth is more effective for specific values of k;
- on real world QBFs, trivial truth does not improve the performance, while backjumping enables QUBE to cope with some instances which could not be solved otherwise.

All the aforementioned issues are the subject of the next three sections. In particular, Section 2 is devoted to the preliminaries, which include a formal presentation of QBFs; a presentation of QUBE decision algorithm(s), including a high-level description of backjumping and trivial truth; and a presentation of the testing methodology. Section 3 is devoted

* We wish to thank Marco Cadoli, Rainer Feldmann, Theodor Lettman, Jussi Rintanen, Marco Schaerf, Stefan Schamberger, for providing us with their systems and helping us to figure them out during our experimental analisys.

to the experimental evaluation of backjumping and trivial truth. In Section 4 we experiment with QUBE, QKN [8], EVALUATE [1], DECIDE [7], and QSOLVE [5] in order to assess the impact of QUBE's optimizations given the current state of the art in QBF solvers.

2 Preliminaries

Consider a set P of propositional letters. An *atom* is an element of P. A *literal* is an atom or the negation of an atom. For each literal l, (i) $\bar{l}$ is x if $l = \neg x$, and is $\neg x$ if $l = x$; (ii) $|l|$ is the atom occurring in l. A *clause* C is a p-ary $(p \geq 0)$ disjunction of literals such that no atom occurs twice in C. A *propositional formula* is an m-ary $(m \geq 0)$ conjunction of clauses. As customary, we represent a clause as a set of literals, and a propositional formula as a set of clauses.

A *QBF* is an expression of the form

$$Q_1 x_1 \ldots Q_n x_n \Phi, \qquad (n \geq 0) \tag{1}$$

where every Q_i $(1 \leq i \leq n)$ is a quantifier, either existential $\exists$ or universal $\forall$; $x_1, \ldots, x_n$ are pairwise distinct atoms in P; and Φ is a propositional formula in the atoms $x_1, \ldots, x_n$. $Q_1 x_1 \ldots Q_n x_n$ is the *prefix* and Φ is the (quantifier-free) *matrix* of (1).

Consider a QBF of the form (1). A literal l occurring in Φ is:

- *existential* if $\exists |l|$ belongs to the prefix of (1), and is *universal* otherwise.
- *unit* in (1) if l is existential, and, for some $k \geq 0$,
  - a clause $\{l, l_1, \ldots, l_k\}$ belongs to Φ, and
  - each expression $\forall |l_i|$ $(1 \leq i \leq k)$ is at the right of $\exists |l|$ in the prefix of (1).
- *monotone* if either l is existential, l occurs in Φ, and $\bar{l}$ does not occur in Φ; or l is universal, l does not occur in Φ, and $\bar{l}$ occurs in Φ.

A clause C is *contradictory* if no existential literal belongs to C. For example, the empty clause (i.e., the clause with no literals) is contradictory.

The semantics of a QBF φ can be defined recursively as follows. If φ contains a contradictory clause then φ is not satisfiable. If the matrix of φ is empty then φ is satisfiable. If φ is $\exists x \psi$ (resp. $\forall x \psi$), φ is satisfiable if and only if φ_x or (resp. and) $\varphi_{\neg x}$ are satisfiable. If $\varphi = Qx\psi$ is a QBF and l is a literal, φ_l is the QBF obtained from ψ by deleting the clauses in which l occurs, and removing $\bar{l}$ from the others. It is easy to see that if φ is a QBF without universal quantifiers, the problem of deciding φ satisfiability reduces to propositional satisfiability (SAT).

Notice that we allow only for propositional formulas in conjunctive normal form (CNF) as matrices of QBFs. Indeed, by applying standard CNF transformations (see, e.g., [9]) it is always possible to rewrite a QBF into an equisatisfiable one satisfying our restrictions.

2.1 QUBE

QUBE is implemented in C on top of SIM [10], an efficient library for propositional satisfiability developed by our group. A C-like high-level description of QUBE is shown in Figure 1. Inspection of QUBE's main routine (*QubeSolver* in Figure 1) reveals that the solver works in the usual way, trying to simplify φ as much as possible (line 26) before choosing how to branch on truth values (line 27), or backtracking to some previous choice point (line 28) whenever a contradictory clause or an empty formula is reached. Clearly, the order in which *ChooseLiteral* picks its choice depends on how literals are arranged in the prefix of φ. Outermost literals are selected first, unless they appear in a contiguous sequence

```
1  φ = ⟨the input QBF⟩;
2  Stack = ⟨the empty stack⟩;

3  function Simplify() {
4   do {
5    φ' = φ;
6    if (⟨a contradictory clause is in φ⟩)
7     return FALSE;
8    if (⟨the matrix of φ is empty⟩)
9     return TRUE;
10   if (⟨l is unit in φ⟩)
11    { |l|.mode = UNIT; Extend(l); }
12   if (⟨l is monotone in φ⟩)
13    { |l|.mode = PURE; Extend(l); }
14  } while (φ' != φ);
15  return UNDEF; }

16 function Backtrack(res) {
17  while (⟨Stack is not empty⟩) {
18   l = Retract();
19   if ((|l|.mode == L-SPLIT) &&
20       ((res == FALSE && |l|.type == ∃) ||
21        (res == TRUE && |l|.type == ∀)))
22     {|l|.mode = R-SPLIT; return l̄; }}
23  return NULL; }

24 function QubeSolver() {
25  do {
26   res = Simplify();
27   if (res == UNDEF) l = ChooseLiteral();
28   else  l = Backtrack(res);
29   if (l != NULL)  Extend(l);
30  } while (l != NULL);
31  return res; }
```

Fig. 1. The pseudo-code of QUBE.

of literals under the scope of the same kind of quantifier: in this case we are free to instantiate the literals in such a sequence with an arbitrary order. In Figure 1 we used the following conventions and notations:

- φ is a global variable initially set to the input QBF.
- *Stack* is a global variable storing the search stack, and is initially empty.
- $\exists$, $\forall$, FALSE, TRUE, UNDEF, NULL, UNIT, PURE, L-SPLIT, R-SPLIT are pairwise distinct constants.
- for each atom x in the input QBF, (i) $x.mode$ is a variable whose possible values are UNIT, PURE, L-SPLIT, R-SPLIT, and (ii) $x.type$ is $\exists$ if x is existential, and $\forall$ otherwise.

QUBE relies on a set of primitives on φ (and *Stack*) to solve the input QBF. The basic services have the purpose of assigning a truth value to a literal and of undoing such assignments. In particular:

- *Extend*(l) first pushes l and φ in the stack; then deletes the clauses of φ in which l occurs, and removes $\bar{l}$ from the others.
- *Retract*() pops the literal and the corresponding QBF that are on top of the stack: the literal is returned, while the QBF is assigned to φ.

Extend() and *Retract*() are thus the finest grained operations that can be performed on QUBE data structure. At a coarser level, QUBE relies on:

- *Simplify*() (detailed in Figure 1) simplifies φ till a contradictory clause is generated (line 6), or the matrix of φ is empty (line 8), or no simplification involving unit clauses and monotone literals is possible (lines 5, 14).
- *ChooseLiteral*() returns a literal l occurring in φ such that for each atom x occurring to the left of $|l|$ in the prefix of φ, x does not occur in φ, or x is existential iff l is existential. *ChooseLiteral*() also sets $|l|.mode$ to L-SPLIT.
- *Backtrack*(*res*) (detailed in Figure 1) pops all the literals and the corresponding QBFs (line 18) from the stack, till a literal l is reached such that $|l|.mode$ is L-SPLIT (line 19), and either (i) l is existential and *res* = FALSE (line 20); or (ii) l is universal and *res* = TRUE (line 21). If such a literal l exists, $|l|.mode$ is set to R-SPLIT, and $\bar{l}$ is returned (line 22). If no such literal exists, NULL is returned (line 23).

The trivial truth optimization is implemented in QuBE as follows. After the simplifications following the branch on an universal variable have been performed, QuBE checks whether the formula obtained from φ by deleting universal literals is satisfiable. If it is, then φ is satisfiable [2]. This optimization can produce dramatic speed-ups, particularly on randomly generated QBFs (see, e.g. [2]). Notice that ours is an optimized version of trivial truth as described in [2], where the check is performed at each branching node. In our version, such a check is not performed after branches on existential variables.

Backjumping comes into play when QuBE reaches a leaf node in the search tree, be it a contradictory clause or an empty formula. At this point, QuBE builds a reason for the current status, i.e., the set of literals directly causing it; then, while bactracking, QuBE refines such reason and tries to understand the branches that indirectly determined the satisfiability status in the leaf. In this way, QuBE can skip the other side of branches on existential literals which would lead again to a contradictory clause, and the other side of branches on universal literals that would lead again to an empty formula. Therefore, the procedure that we implemented in QuBE is a generalization of conflict-direct backjumping as it is usually implemented in SAT solvers. It turns out that, exactly as in SAT, backjumping has some overhead, but it can produce significant speed-ups [3].

Currently QuBE is distributed in two fashions: QuBE-BT, featuring standard chronological backtracking, and QuBE-BJ featuring non-chronological, conflict-directed backjumping. The choice between these two versions determines whether we are running the backjumping optimization or not. As for the trivial truth optimization, it is performed by default in both versions: we denote with "QuBE-BT -tt" and "QuBE-BJ -tt" the procedures corresponding, respectively, to QuBE-BT and QuBE-BJ with trivial truth turned off. QuBE's options other than backjumping and trivial truth are set according to their default values. As matter of fact, all the experiments have been run with the same version of *ChooseLiteral*, i.e., Böhm's heuristic [11] for SAT, modified as in [5] to consider existential literals only. Other configurations of QuBE, including those that we do not take into account here, are described in [4] to which we refer the reader for further details.

2.2 Testing Methodology

To experiment with various settings of QuBE and to compare it with other state-of-the-art solvers,[1] we have considered sets of random QBFs generated according to the "model A" as described in [6]. With this model, each QBF has the following 4 properties: (i) the prefix consists of k sequences, each sequence has n quantifiers, and each two quantifiers in a same sequence, are of the same type, (ii) the rightmost quantifier is $\exists$, (iii) the matrix consists of l clauses, (iv) each clause consists of h literals of which at least 2 are existential. In our experiments we fix $h = 5$ and we run the solvers with (some of) the settings $k = 2, 3, 4, 5$ and $n = 50, 100, 150$, in order to understand how the effectiveness of backjumping and trivial truth is related to a change of those parameters. The choice of $h = 5$ is motivated by the analysis presented in [6]: here the authors show that "model A" instances with $h = 5$ turn out to be challenging on average. On the other hand, smaller values of h are not quite as good: for instance, according to [6], in the case of $h = 3$ only a few samples turn out to be really difficult for the current state-of-the-art solvers.

For every problem, i.e., every specific value of k and n, the range of l is experimentally chosen to cover at least the 100%-satisfiable - 100% unsatisfiable transition. In order to meet this requirement and to compare performances among different settings, we fixed $1 \leq l/n \leq 20$ for all the problems. Given a point, i.e., a specific value of k and l/n, we generate

[1] Experiments run on several identical Pentium III, 600MHz, 128MBRAM equipped with Linux Suse 6.x, kernel release 2.2.14. QuBE is compiled with GNU C release 2.95.2.

100 samples; for each sample and for all the solvers we measure the running time in CPU seconds. For QuBE, we also count the number of times that *ChooseLiteral* is called, i.e., the number of nodes in the search tree. Notice that when trivial truth is enabled, this includes the nodes explored by QuBE when running the "look-ahead" search. In order to save time, the execution of a solver on a sample is timed out after 1200s. Even more, for any point $k, l/n$, the execution of a solver is stopped after it exceeds the timeout on more than 50% of the samples, or:

- up to 6 were attempted so far and at least 5 timed out, or
- up to 10 were attempted so far and at least 8 of them timed out, or
- up to 20 were attempted so far and at least 13 of them timed out.

In the following, we say that a point cannot be solved whenever any of the above conditions is met. For each point we consider the Q*-percentiles* for $Q = 50, 60, 70, 80, 90, 100$, the Q-percentile being the value such that $Q\%$ of the observations falls at or below it [12]. Our main statistical tool is the 50-percentile – best known as the *median* value – which we use as a resistant estimate for the average behaviour of the solvers on each point. The percentiles for $Q \geq 60$ serve the purpose of showing the behaviour on the hardest instances: in particular the 100-percentile corresponds to the worst case behaviour of the solvers on each point. When a point cannot be solved, the values of all the Q-percentiles default to 1200s.

Our experimental analysis includes the 38 problems contributed by Rintanen in [7].[2] The problems are translations from planning problems into the language of QBFs. They are interesting because:

- none of them can be solved by all the solvers at hand, and
- they share some "structure" with the original planning problem.

The second point is particularly relevant when comparing results on Rintanen's problems with results on "model A" instances. In SAT (see, e.g., [10]) it is often the case that the performance of some heuristics and optimizations is quite different when changing from randomly generated instances to real world ones. Therefore, although there is not yet any evidence in this direction for QBF, we included real world instances to get a wide-spectrum evaluation of QuBE and other state-of-the-art solvers.

3 Backjumping vs. Trivial Truth

We start the review of our experimental results with the comparison showed in Figure 2 among the four versions of QuBE giving all possible combinations of backjumping and trivial truth, including QuBE-BT -tt, i.e, QuBE's "vanilla" algorithm. The plots are arranged in two columns, running times are reported on the left and nodes are reported on the right; going from top to bottom, we show the results of problems $k = 2, 3, 4, 5$ with $n = 50$. In every plot, the x-axis is the range of l/n, and the y-axis is either the running time or the number of nodes, both in logarithmic scale. The percentage of satisfiable samples is represented by the dotted curve in the background. Notice that in the plots we omit the points which could not be solved.

By looking at the plots in Figure 2 and focusing on the running times for $k = 2$ (top-left), we immediatly notice that trivial truth causes at least 3 orders of magnitude speed-up for the hardest formulas, i.e., those with l/n = 4. As a matter of fact, QuBE-BT median running time peaks at slightly less than 1s, while QuBE-BT -tt cannot solve more than 50%

[2] Available at `www.informatik.uni-freiburg.de/~rintanen/qbf.html`.

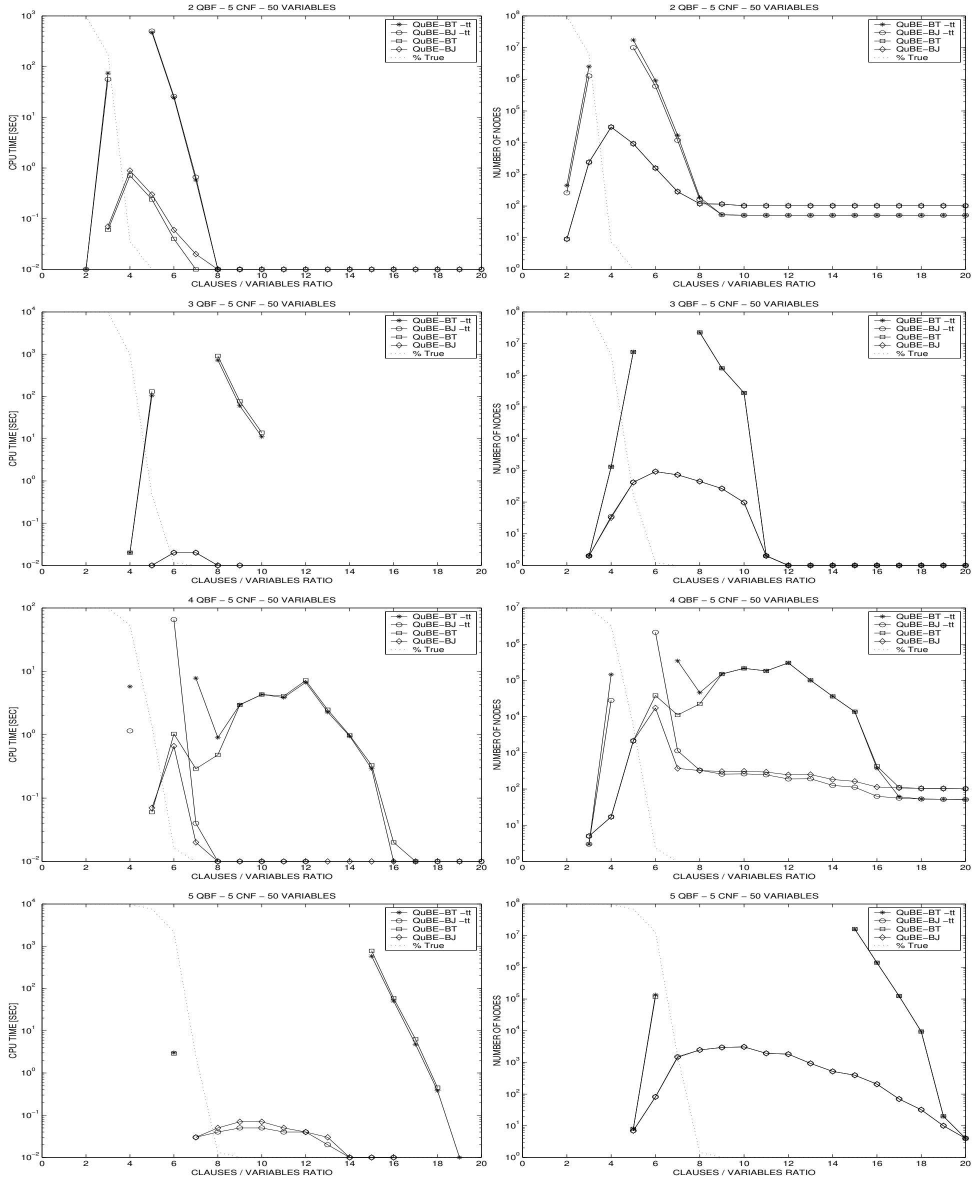

Fig. 2. QUBE median running time (left) and number of nodes (right) with $n = 50$, $k = 2, 3, 4, 5$.

of the samples within the time limit. Still with $k = 2$, but looking at the number of nodes (Fig. 2 top-right), we see that the good performance of trivial truth is readily explained by a similar order-of-magnitude difference in the number of nodes: the difference is even more impressing if we consider that such number includes the nodes explored by trivial truth. On the other hand, backjumping does not seem to pay off: although the median number of nodes for QUBE-BJ (resp. QUBE-BJ -tt) is always less than or equal to the same statistic for QUBE-BT (resp. QUBE-BJ -tt), the difference is not enough to pay off the additional overhead. Indeed, it turns out that backjumping alone (QUBE-BT -tt) is not able to decrease the median running time of QUBE below the 1200s threshold for $l/n = 4$.

		QuBE-BT -tt	QuBE-BJ -tt	QuBE-BT	QuBE-BJ
Test File	Value	Time	Time	Time	Time
B..3ii.4.3	0	-	5.690	-	7.900
B..3ii.5.2	0	-	97.450	-	128.460
B..3iii.4	0	-	1.630	-	2.540
C..21v.22	1	140.630	163.950	-	-
C..22v.23	1	292.900	341.330	-	-
C..23v.24	1	605.050	701.300	-	-
T..6.1.iv.11	0	32.500	6.940	40.220	8.460
T..6.1.iv.12	1	14.610	1.830	19.330	2.330
T..7.1.iv.13	0	753.020	139.720	941.300	144.280
T..7.1.iv.14	1	311.830	28.910	416.700	36.090

Table 1. QUBE optimizations on (some) real world instances.

The situation is quite different for all values $k \geq 3$. Indeed, trivial truth has a much more limited impact, in particular for $k = 3$ (Fig. 2 second row) and $k = 5$ (Fig. 2 bottom row), while backjumping is delivering order-of-magnitude speed-ups on the same settings. This is utterly evident by looking at the running times for $k = 5$ (Fig. 2 bottom-left), where QUBE-BJ and QUBE-BJ -tt curves both peak at less than 0.1s, while QUBE-BT and QUBE-BT -tt both peak – possibly well – above the timeout threshold. Again, it is a similarly large reduction in the search space size (Fig. 2 bottom-right) that determines the difference in performance.

The plots with $k = 4$ (Fig. 2 third row) are even more interesting, since trivial truth and backjumping show to be incomparable even considering a single setting of k and l/n. Neither backjumping alone (QUBE-BJ -tt) nor trivial truth alone (QUBE-BT) are able to achieve optimum performances for $k = 4$. Looking at running times (left) we see that in the first part of the plot, up to $l/n = 6$, QUBE-BT is faster than QUBE-BJ -tt, but the situation is reversed in the second half of the plot, with the two curves rejoicing only for $l/n \geq 17$. The situation is similar when looking at the number of nodes (right), with the additional remark that QUBE-BJ (resp. QUBE-BJ -tt) always explores less search space than QUBE-BT (resp. QUBE-BT -tt). Interestingly, QUBE-BT median running time does not exceed the timeout threshold, while QUBE-BJ -tt peaks above it; nevertheless, the two optimizations can interact positively, since QUBE-BJ is able to solve all the points.

In Table 1 we show the results obtained by running QUBE on some of the real world samples that we discussed in Subsection 2.2. In particular we omit those instances (there are 28 of them) in which QUBE is either able (11) or not able (17) to solve all the instances, no matter what optimizations are tried. The remaning 10 instances are shown in Table 1, divided in three classes: “blocks-world” (“B” instances), “chain” (“C” instances), and “bomb in the toilet” problems (“T” instances). In the columns of Table 1, besides the satisfiability status, we report for each version of QUBE the running time in seconds: instances exceeding the time limit are denoted with a dash. In the case of “B” instances, backjumping is the optimization of choice: QUBE-BJ -tt as well as QUBE-BJ are able to solve all the three instances, while QUBE-BT -tt and QUBE-BT are not. We also notice that trivial truth is causing a performance degradation in QUBE-BJ which turns out to be slower than QUBE-BJ -tt. With “C” instances, even QUBE’s vanilla version is able to solve the corresponding QBFs, while backjumping does not seem to pay off. Noticeably, the versions featuring trivial truth are not able to solve the instances within the time limit. With “T” instances, QUBE-BJ -tt is again the fastest on all the problems, thus confirming the effectiveness of backjumping.

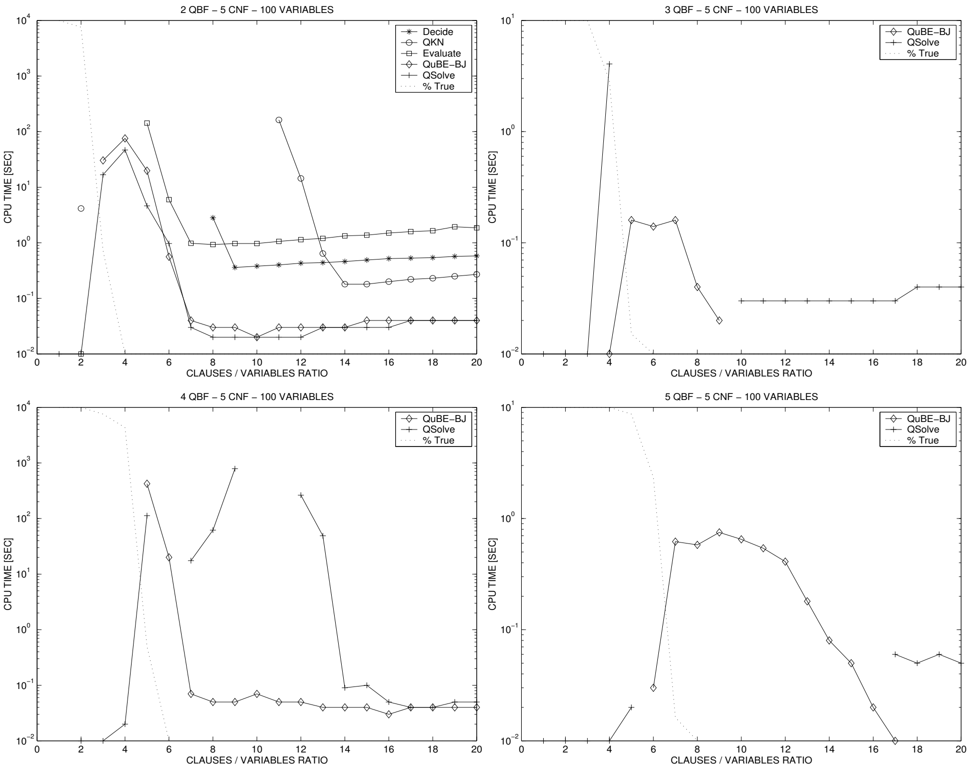

Fig. 3. QUBE and state-of-the-art solvers on $k = 2, 3, 4, 5$ and $n = 100$ problems.

4 QUBE and state-of-the-art QBF solvers

In Figure 3 we review the experiments comparing state-of-the-art solvers with QUBE. Going from left to right, top to bottom, we show the problems $k = 2, 3, 4, 5$ with $n = 100$. The plots are arranged in the same way as in Figure 2. We run QUBE-BJ and QSOLVE [5] with all the settings, while for $k = 2$ we also tested DECIDE [7], EVALUATE [1] and QKN [8]. Looking at Figure 3 (top left) we immediatly see that QUBE and QSOLVE perform roughly the same (with QSOLVE being slightly better than QUBE) and better than all the other solvers that we tested: EVALUATE, QKN and DECIDE all fail to solve the hardest points. Still in Figure 3, for $k \geq 3$ we further observe that QUBE is always faster than QSOLVE, sometimes by orders of magnitude. Since QSOLVE runs trivial truth (and trivial falsity), but no backjumping, we take this as further evidence on the effectiveness of backjumping for values of $k \geq 3$. In particular, we remark the qualitative similarity between QSOLVE with $k = 4$ and $n = 100$ in Figure 3 and QUBE-BT with $k = 4$ and $n = 50$ in Figure 2, and the fact that all the considerations that we did on $n = 50$ seems to carry over to $n = 100$.

We conclude the comparison between QUBE and the other state-of-the-art solvers with the results presented in Table 2. In the Table, we report the running times of QUBE and other solvers on 33 out of 38 problems: the remaining instances could not be solved by any of the systems. As we can see from Table 2, the best solver overall turns out to be DECIDE with 33 problems solved and 42.28s average running time (on solved samples), followed by QUBE, with 18 problems solved and 73.21s, and QSOLVE with 11 problems solved and 149.29s. QKN and EVALUATE with, respectively, 1 and 0 problems solved, trail the list. In this regard, we point out that DECIDE features “inversion of quantifiers” and “sampling”

		QuBE-BJ -tt	QSolve	Evaluate	QKN	Decide
Test File	Value	Time	Time	Time	Time	Time
B..3i.4.4	0	-	-	-	-	0.11
B..3i.5.3	0	-	-	-	-	49.72
B..3i.5.4	1	-	-	-	-	7.53
B..3ii.4.3	0	5.690	-	-	-	0.07
B..3ii.5.2	0	97.450	-	-	-	0.41
B..3ii.5.3	1	-	-	-	-	1.80
B..3iii.4	0	1.630	-	-	-	0.02
B..3iii.5	1	-	-	-	-	0.56
B..4i.6.4	0	-	-	-	-	3.72
B..4ii.6.3	0	-	-	-	-	3.10
B..4ii.7.2	0	-	-	-	-	8.79
B..4iii.6	0	-	-	-	-	1.51
B..4iii.7	1	-	-	-	-	729.11
C..12v.13	1	0.220	6.45	-	-	0.07
C..13v.14	1	0.470	20.1	-	-	0.10
C..14v.15	1	0.970	63.47	-	-	0.12
C..15v.16	1	2.020	199.31	-	-	0.16
C..16v.17	1	4.200	624.97	-	-	0.19
C..17v.18	1	8.920	-	-	-	0.23
C..18v.19	1	18.480	-	-	-	0.28
C..19v.20	1	38.590	-	-	-	0.31
C..20v.21	1	85.400	-	-	-	0.36
C..21v.22	1	163.950	-	-	-	0.40
C..22v.23	1	341.330	-	-	-	0.48
C..23v.24	1	701.300	-	-	-	0.56
T..10.1.iv.20	1	-	-	-	-	6.30
T..16.1.iv.32	1	-	-	-	-	43.15
T..2.1.iv.3	0	0.000	0	-	0.147	0.01
T..2.1.iv.4	1	0.000	0	-	-	0.02
T..6.1.iv.11	0	6.940	11.81	-	-	36.17
T..6.1.iv.12	1	1.830	8.53	-	-	0.91
T..7.1.iv.13	0	139.720	426.93	-	-	428.20
T..7.1.iv.14	1	28.910	203.53	-	-	1.64

Table 2. QUBE and state-of-the-art solvers on (some) real world instances.

mechanisms which are particularly effective on these benchmarks. Athough QUBE does not feature such mechanisms its performances are quite good on these tests: on “C” instances this is entirely due to its heuristics and efficient implementation, while backjumping turns out to be the crucial factor in the solution of “B” and “T” instances.

5 Conclusions

Summing up, the results of our experimental evaluation of backjumping and trivial truth, largely confirm that they are of incomparable strengths, but they both help in obtaining substantial speed-ups. In more details:

– on random kQBFs, backjumping is more effective with $k \geq 3$ and, in particular, with $k = 2i + 1$; trivial truth is very effective on $k = 2$ and less effective on $k \geq 3$. $k = 4$

is an example where, on the same setting of k and n, trivial truth and backjumping show a different behaviour, as well as the ability to interact positively. On this class of problems they can both be used to cope with instances that are hardly solved by using them separately;

– on real world QBFs, trivial truth does not improve the performances, while backjumping enables QUBE to solve some "blocks world" and "bomb in the toilet" instances which could not be solved otherwise.

Thanks to its optimizations, QUBE turns out to be very competitive with state-of-the-art solvers on "model A" instances for all values of k and n that we tried, but in particular for $k = 3, 4, 5$ and $n = 100$. In the case of real world samples, QUBE with backjumping comes second best after DECIDE which is indeed highly tuned to solve those problems.

References

1. M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proc. of AAAI*, 1998.
2. M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An Algorithm to Evaluate Quantified Boolean Formulae and its Experimental Evaluation. In *Highlights of Satisfiability Research in the Year 2000*. IOS Press, 2000.
3. E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic Satisfiability. In *Proc. of IJCAI*, 2001. To appear.
4. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE: a system for Deciding Quantified Boolean Formulas Satisfiability. In *Proc. of IJCAR*, 2001.
5. R. Feldmann, B. Monien, and S. Schamberger. A Distributed Algorithm to Evaluate Quantified Boolean Formulae. In *Proc. of AAAI*, 2000.
6. Ian Gent and Toby Walsh. Beyond NP: the QSAT phase transition. In *Proc. of AAAI*, pages 648–653, 1999.
7. J. T. Rintanen. Improvements to the Evaluation of Quantified Boolean Formulae. In *Proc. of IJCAI*, pages 1192–1197, 1999.
8. H. Kleine-Büning and M. Karpinski and A. Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
9. D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.
10. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of IJCAR*, LNCS. Springer Verlag, 2001. To appear.
11. M. Buro and H. Kleine-Büning. Report on a SAT competition. Technical Report 110, University of Paderborn, November 1992.
12. D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman and Co., 1993.