Backjumping for Quantified Boolean Logic Satisfiability

Enrico Giunchiglia and **Massimo Narizzano** and **Armando Tacchella**
DIST - Università di Genova
Viale Causa 13, 16145 Genova, Italy
{enrico,mox,tac}@mrg.dist.unige.it

Abstract

The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important research issue in Artificial Intelligence. Many decision procedures have been proposed in the last few years, most of them based on the Davis, Logemann, Loveland procedure (DLL) for propositional satisfiability (SAT). In this paper we show how it is possible to extend the conflict-directed backjumping schema for SAT to QBF: when applicable, it allows to jump over existentially quantified literals while backtracking. We introduce solution-directed backjumping, which allows the same for universally quantified literals. Then, we show how it is possible to incorporate both conflict-directed and solution-directed backjumping in a DLL-based decision procedure for QBF satisfiability. We also implement and test the procedure: The experimental analysis shows that, because of backjumping, significant speed-ups can be obtained. While there have been several proposals for backjumping in SAT, this is the first time –as far as we know– this idea has been proposed, implemented and experimented for QBFs.

1 Introduction

The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important research issue in Artificial Intelligence. Many reasoning tasks involving abduction, reasoning about knowledge, non monotonic reasoning, are PSPACE-complete reasoning problems and are reducible in polynomial time to the problem of determining the satisfaction of a QBF. More important, since QBF reasoning is the prototypical PSPACE problem, many of these reductions are readily available. For these reasons, we have seen in the last few years the presentation of several implemented decision procedures for QBFs, like QKN [Kleine-Büning, H. and Karpinski, M. and Flögel, A., 1995], EVALUATE [Cadoli *et al.*, 1998, Cadoli *et al.*, 2000], DECIDE [Rintanen, 1999b], QUIP [Egly *et al.*, 2000], QSOLVE [Feldmann *et al.*, 2000]. Most of the above decision procedures are based on the Davis, Logemann, Loveland procedure (DLL) for propositional satisfiability [Davis *et al.*, 1962] (SAT). This is because it is rather easy to extend DLL to deal with QBFs, and also because DLL is at the core of many state-of-the-art deciders for SAT.

In this paper we show how it is possible to extend the conflict-directed backjumping schema for SAT (see, e.g., [Prosser, 1993, Bayardo, Jr. and Schrag, 1997]) to QBF: when applicable, it allows to jump over existentially quantified literals while backtracking. We introduce solution-directed backjumping, which allows the same for universally quantified literals. Then, we show how it is possible to incorporate both conflict-directed and solution-directed backjumping in a DLL-based decision procedure for QBF satisfiability. We also implement and test the procedure: The experimental analysis shows that, because of backjumping, significant speed-ups can be obtained. While there have been several proposals for backjumping in SAT, this is the first time –as far as we know– this idea has been proposed, implemented and experimented for QBFs.

The paper is structured as follows. In Section 2 we introduce some formal preliminaries necessary for the rest of the paper. In Section 3 we present QUBE, a DLL based decision procedure for QBFs. Section 4 is devoted to the presentation of the theoretical results at the basis of the backjumping procedure presented in Section 5. The experimental analysis is reported in Section 6. We end the paper with the conclusions and future work in Section 7. Proofs are omitted for lack of space.

2 Formal 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. In the following, for any literal l,

- $|l|$ is the atom occurring in l; and
- $\bar{l}$ is $\neg l$ if l is an atom, and is $|l|$ otherwise.

A *clause* C is an n-ary ($n \geq 0$) disjunction of literals such that, for any two distinct disjuncts l, l' in C, it is not the case that $|l| = |l'|$. A *propositional formula* is a k-ary ($k \geq 0$) conjunction of clauses. As customary in SAT, we represent a clause as a set of literals, and a propositional formula as a set of clauses. With this notation, e.g.,

- the clause $\{\}$ is the *empty clause* and stands for the empty disjunction,
- the propositional formula $\{\}$ is the *empty set of clauses* and stands for the empty conjunction,
- the propositional formula $\{\{\}\}$ stands for the set of clauses whose only element is the empty clause.

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 *matrix* of (1).[1] For example, the expression

$$\begin{aligned} \forall x_1 \exists x_2 \forall x_3 \exists x_4 \exists x_5 \{ & \{x_1, x_3, x_4\}, \{x_1, \neg x_3, \neg x_4, \neg x_5\}, \\ & \{x_1, \neg x_4, x_5\}, \{\neg x_1, x_2, x_5\}, \\ & \{\neg x_1, x_3, x_4\}, \{\neg x_1, x_3, \neg x_4\}, \\ & \{\neg x_1, \neg x_2, \neg x_3, \neg x_5\}\} \end{aligned} \tag{2}$$

is a QBF with 7 clauses.

Consider a QBF (1). A literal l 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 $m \geq 0$,
  - a clause $\{l, l_1, \ldots, l_m\}$ belongs to Φ, and
  - each expression $\forall |l_i|$ $(1 \leq i \leq m)$ occurs 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 is contradictory.

The semantics of a QBF φ can be defined recursively as follows. If φ contains a contradictory clause then φ is FALSE. If the matrix of φ is the empty set of clauses then φ is TRUE. If φ is $\exists x \psi$ (respectively $\forall x \psi$), φ is TRUE if and only if φ_x or (respectively and) $\varphi_{\neg x}$ are TRUE. 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 SAT.

3 QUBE

QUBE is implemented in C on top of SIM, an efficient decider for SAT developed by our group [Giunchiglia *et al.*, 2001a]. A high-level description of QUBE is presented in Figure 1.[2] In Figure 1,

- φ is a global variable initially set to the input QBF.

```
1  φ := ⟨the input QBF⟩;   Stack := ⟨the empty stack⟩;
2  function Simplify()
3   do
4     φ' := φ;
5     if (⟨a contradictory clause is in φ⟩) return FALSE;
6     if (⟨the matrix of φ is empty⟩) return TRUE;
7     if (⟨l is unit in φ⟩)
8       |l|.mode := UNIT; Extend(l);
9     if (⟨l is monotone in φ⟩)
10      |l|.mode := PURE; Extend(l);
11  while (φ' ≠ φ);
12  return UNDEF;

13 function Backtrack(res)
14  while (⟨Stack is not empty⟩)
15    l := Retract();
16    if (|l|.mode = L-SPLIT) and
17        ((res = FALSE and |l|.type = ∃) or
18         (res = TRUE and |l|.type = ∀))
19          |l|.mode := R-SPLIT; return l̄;
20  return NULL;

21 function QubeSolver()
22  do
23    res := Simplify();
24    if (res = UNDEF)  l := ChooseLiteral();
25    else  l := Backtrack(res);
26    if (l ≠ NULL)  Extend(l);
27  while (l ≠ NULL);
28  return res;
```

Figure 1: The algorithm of QUBE.

- *Stack* is a global variable storing the search stack, and is initially empty.
- FALSE, TRUE, UNDEF, NULL, UNIT, PURE, L-SPLIT, R-SPLIT are pairwise distinct constants.
- for each atom x in the input QBF,
  - $x.mode$ is a property of x whose possible values are UNIT, PURE, L-SPLIT, R-SPLIT, and have the obvious meaning, and
  - $x.type$ is $\exists$ if x is existential, and $\forall$ otherwise.
- *Extend*(l) deletes the clauses of φ in which l occurs, and removes $\bar{l}$ from the others. Additionally, before performing the above operations, pushes l and φ in the stack.
- *Retract*() pops the literal and corresponding QBF that are on top of the stack: the literal is returned, while the QBF is assigned to φ. (Intuitively, *Retract* is the "inverse" operation of *Extend*).
- *Simplify*() simplifies φ till a contradictory clause is generated (line 5), or the matrix of φ is empty (line 6),

[1] Traditionally, the syntax of QBFs allows for arbitrary propositional formulas as matrices. However, the problem of determining the satisfiability of a QBF whose matrix is a set of clauses remains PSPACE complete.

[2] We use the following pseudocode conventions. Indentation indicates block structure. Two instructions on the same line belong to the same block. ":=" is the assignment operator. The constructs **while** ⟨*cond*⟩ ⟨*block*⟩, **do** ⟨*block*⟩ **while** ⟨*cond*⟩, **if**⟨*cond*⟩ ⟨*block*$_1$⟩ **else** ⟨*block*$_2$⟩ have the same interpretation as in the C language.

or no simplification is possible (lines 4, 11). The simplifications performed in lines 8 and 10 correspond to Lemmas 6, 4-5 respectively of [Cadoli *et al.*, 1998, Cadoli *et al.*, 2000].

- *ChooseLiteral*() returns a literal l occurring in φ such that for each atom x occurring to the left of $|l|$ in the prefix of the input QBF,
  - x and $\neg x$ do not occur in φ, or
  - x is existential iff l is existential.

  ChooseLiteral() also sets $|l|.mode$ to L-SPLIT.

- *Backtrack*(*res*): pops all the literals and corresponding QBFs (line 15) from the stack, till a literal l is reached such that $|l|.mode$ is L-SPLIT (line 16), and either
  - l is existential and *res* = FALSE (line 17); or
  - l is universal and *res* = TRUE (line 18).

  If such a literal l exists, $|l|.mode$ is set to R-SPLIT, and $\bar{l}$ is returned (line 19). If no such literal exists, NULL is returned (line 20).

It is easy to see that QUBE is a generalization of DLL: QUBE and DLL have the same behavior on QBFs without universal quantifiers.

To understand QUBE behavior, consider the QBF (2). For simplicity, assume that *ChooseLiteral* returns the negation of the first atom in the prefix which occurs in the matrix of the QBF under consideration. Then, the tree searched by QUBE when φ is (2) is represented in Figure 2. In Figure 2, each node shows

- the sequence of literals assigned by QUBE before a branch takes place (first line): for each literal l in the sequence, we also show the value of $|l|.mode$; and
- the matrix of the resulting QBF, prefixed by a label (second line).

As the result of the computation, QUBE would correctly return FALSE, i.e., (2) is unsatisfiable.

4 Backjumping

Let φ be a QBF (1). Consider φ.

In the following, for any finite sequence $\mu = l_1; \ldots; l_m$ ($m \geq 0$) of literals, we write

- *Assign*(μ, Φ) as an abbreviation for

$$Assign(l_m, \ldots, (Assign(l_1, \Phi)) \ldots),$$

  where, if Ψ is a set of clauses, *Assign*(l, Ψ) is the set of clauses obtained from Ψ by deleting the clauses in which l occurs, and removing $\bar{l}$ from the others.

- $[Q_1 x_1 \ldots Q_n x_n]_\mu$ as an abbreviation for the expression obtained from $Q_1 x_1 \ldots Q_n x_n$ by removing $Q_i x_i$ whenever x_i or $\neg x_i$ is in μ.

- φ_μ as an abbreviation for the QBF

$$[Q_1 x_1 \ldots Q_n x_n]_\mu Assign(\mu, \Phi).$$

Intuitively, if μ is a sequence of literals representing an assignment, φ_μ is the QBF resulting after the literals in μ are assigned. For example, considering Figure 2, if φ is (2), Φ is $(F4)$, and μ is $\neg x_1; x_3$, then φ_μ is $\exists x_2 \exists x_4 \exists x_5 \Phi$.

As in constraint satisfaction (see, e.g., [Dechter, 1990, Prosser, 1993, Bayardo, Jr. and Schrag, 1997]) it may be the case that only a subset of the literals in the current assignment is responsible for the result (either TRUE or FALSE) of φ_μ satisfiability. Then, assuming that it is possible to effectively determine such a subset ν, we could avoid doing a right branch on a literal l, if l is not in ν. To make these notions precise we need the following definitions.

A finite sequence $\mu = l_1; \ldots; l_m$ ($m \geq 0$) of literals is an *assignment for* φ if for each literal l_i in μ

- l_i is unit, or monotone in $\varphi_{l_1;\ldots;l_{i-1}}$; or
- l_i occurs in $\varphi_{l_1;\ldots;l_{i-1}}$ and for each atom x occurring to the left of $|l_i|$ in the prefix of the input QBF,
  - x and $\neg x$ do not occur in $\varphi_{l_1;\ldots;l_{i-1}}$, or
  - x is existential iff l_i is existential.

Consider an assignment $\mu = l_1; \ldots; l_m$ for φ.
A set of literals ν is a *reason for* φ_μ *result* if

- $\nu \subseteq \{l_1, \ldots, l_m\}$, and
- for any assignment μ' for φ such that
  - $\nu \subseteq \{l : l \text{ is in } \mu'\}$,
  - $\{|l| : l \text{ is in } \mu'\} = \{|l| : l \text{ is in } \mu\}$,

  $\varphi_{\mu'}$ is satisfiable iff φ_μ is satisfiable.

For example, if φ is (2),

- if μ is $x_1; \neg x_2$, then $\{x_1\}$ is a reason for φ_μ result: For each assignment $\mu' \in \{x_1; \neg x_2, x_1; x_2\}$, $\varphi_{\mu'}$ is unsatisfiable, and
- if μ is $\neg x_1; \neg x_3$, then $\{\neg x_1\}$ is a reason for φ_μ result: For each assignment $\mu' \in \{\neg x_1; \neg x_3, \neg x_1; x_3\}$, $\varphi_{\mu'}$ is satisfiable.

Intuitively,

- when given an assignment μ such that either the matrix φ_μ is empty or it contains a contradictory clause, we first compute a reason ν for φ_μ result, and
- while backtracking, we dynamically modify the reason ν for the current result. Furthermore, we use ν in order to avoid useless branches: In particular, we avoid doing a right branch on a literal l if $\bar{l}$ is not in ν.

The theorems in the next subsections —in which we show how to compute the reason— are an easy consequence of the following proposition.

Let ν be a reason for φ_μ result. We say that

- ν is a *reason for* φ_μ *satisfiability* if φ_μ is satisfiable, and
- ν is a *reason for* φ_μ *unsatisfiability*, otherwise.

Proposition 1 *Let φ be a QBF (1). Let $\mu = l_1; \ldots; l_m$ ($m \geq 0$) be an assignment for φ. Let μ' be a sequence obtained by removing some of the literals in μ. Let ν be the set of literals in μ'.*

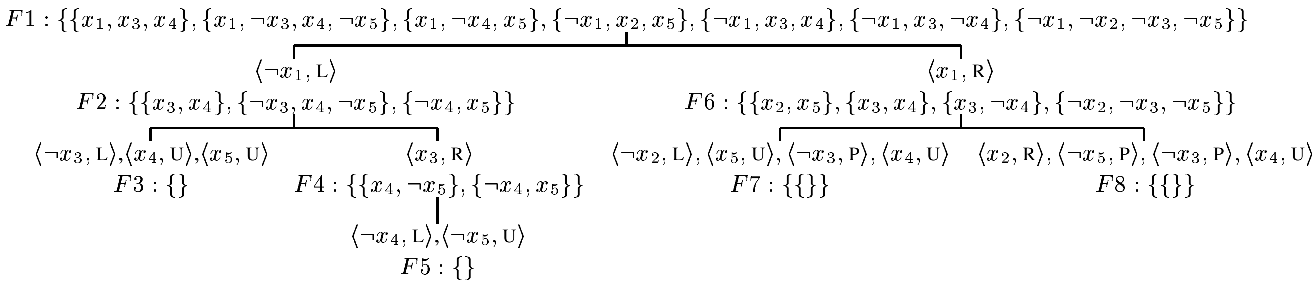

Figure 2: QuBE computation tree for (2). U, P, L, R stand for UNIT, PURE, L-SPLIT, R-SPLIT respectively. The prefix is $\forall x_1 \exists x_2 \forall x_3 \exists x_4 \exists x_5$.

- *ν is a reason for φ_μ unsatisfiability iff the QBF*

$$\exists|l_1| \ldots \exists|l_m|[Q_1 x_1 \ldots Q_n x_n]_\mu \mathit{Assign}(\mu', \Phi)$$

is unsatisfiable.

- *ν is a reason for φ_μ satisfiability iff the QBF*

$$\forall|l_1| \ldots \forall|l_m|[Q_1 x_1 \ldots Q_n x_n]_\mu \mathit{Assign}(\mu', \Phi)$$

is satisfiable.

4.1 Conflict-directed Backjumping

The following Theorem allows us to compute the reason for φ_μ result when the matrix of φ_μ contains a contradictory clause. Given a sequence of literals μ, we say that a literal l is FALSE-*irrelevant* in μ if —assuming μ' is the sequence obtained by deleting l in μ— $\varphi_{\mu'}$ contains a contradictory clause.

Theorem 1 *Let φ be a QBF. Let μ be an assignment for φ such that φ_μ contains a contradictory clause. Let μ' be a sequence obtained by recursively eliminating a FALSE-irrelevant literal in μ. Let ν be the set of literals in μ'. Then ν is a reason for φ_μ unsatisfiability.*

If φ is (2), this Theorem allows us to conclude, e.g., that —with reference to Figure 2—

- if μ is $x_1; \neg x_2; x_5; \neg x_3; x_4$, then $\{x_1, \neg x_3, x_4\}$ is a reason for φ_μ unsatisfiability, and
- if μ is $x_1; \neg x_2; x_5; \neg x_3; \neg x_4$, then $\{x_1, \neg x_3, \neg x_4\}$ is a reason for φ_μ unsatisfiability.

Our next step is to show how it is possible to compute reasons for φ_μ unsatisfiability while backtracking.

Theorem 2 *Let φ be a QBF. Let l be a literal. Let $\mu; l$ be an assignment for φ. Let ν be a reason for $\varphi_{\mu;l}$ unsatisfiability.*

1. *If l is not in ν, then ν is a reason for φ_μ unsatisfiability.*
2. *If $l \in \nu$, and l is universal, then $\nu \setminus \{l\}$ is a reason for φ_μ unsatisfiability.*
3. *If $l \in \nu$, l is existential and monotone in φ_μ, then $\nu \setminus \{l\}$ is a reason for φ_μ unsatisfiability.*
4. *If $l \in \nu$, $\bar{l} \in \nu'$, ν' is a reason for $\varphi_{\mu;\bar{l}}$ unsatisfiability, and l is existential, then $(\nu \cup \nu') \setminus \{l, \bar{l}\}$ is a reason for φ_μ unsatisfiability.*

If φ is (2), considering Theorem 2 and Figure 2:

- given what we said in the paragraph below Theorem 1, the 4th statement allows us to conclude that $\{x_1, \neg x_3\}$ is a reason for φ_μ unsatisfiability when μ is $x_1; \neg x_2; x_5; \neg x_3$,
- from the above, the 2nd statement allows us to conclude that $\{x_1\}$ is a reason for φ_μ unsatisfiability when μ is $x_1; \neg x_2; x_5$,
- from the above, the 1st statement allows us to conclude that $\{x_1\}$ is a reason for φ_μ unsatisfiability when μ is in $\{x_1; \neg x_2, x_1\}$.

From the last item, it follows that looking for assignments satisfying φ_μ when μ begins with x_1, is useless. Given this, our "backjumping" procedure would have avoided the generation of the branch leading to ($F8$) in Figure 2.

4.2 Solution-directed Backjumping

The following Theorem allows us to compute the reason for φ_μ result when the matrix of φ_μ is empty. Given a sequence of literals μ, we say that a literal l is TRUE-*irrelevant* in μ if —assuming μ' is the sequence obtained by deleting l in μ— the matrix of $\varphi_{\mu'}$ is empty.

Theorem 3 *Let φ be a QBF. Let μ be an assignment for φ such that the matrix of φ_μ is empty. Let μ' be a sequence obtained by recursively eliminating a TRUE-irrelevant literal in μ. Let ν be the set of literals in μ'. Then ν is a reason for φ_μ satisfiability.*

With reference to Figure 2, the above Theorem allows us to conclude that, e.g., $\{\neg x_1, x_4, x_5\}$ is a reason for φ_μ satisfiability, if μ is $\neg x_1; \neg x_3; x_4; x_5$ and φ is (2).

Theorem 4 *Let φ be a QBF. Let l be a literal. Let $\mu; l$ be an assignment for φ. Let ν be a reason for $\varphi_{\mu;l}$ satisfiability.*

1. *If l is not in ν, then ν is a reason for φ_μ satisfiability.*
2. *If $l \in \nu$, and l is existential, then $\nu \setminus \{l\}$ is a reason for φ_μ satisfiability.*
3. *If $l \in \nu$, l is universal and monotone in φ_μ, then $\nu \setminus \{l\}$ is a reason for φ_μ satisfiability.*
4. *If $l \in \nu$, $\bar{l} \in \nu'$, ν' is a reason for $\varphi_{\mu;\bar{l}}$ satisfiability, and l is universal, then $(\nu \cup \nu') \setminus \{l, \bar{l}\}$ is a reason for φ_μ satisfiability.*

If φ is (2), considering Theorem 4 and Figure 2:

```
1  function Backjump(res)
2   wr := InitWr(res);
3   while (⟨Stack is not empty⟩)
4     l := Retract();
5     if (l ∈ wr)
6       if (res = FALSE and |l|.type = ∃) or
7          (res = TRUE and |l|.type = ∀)
8         if (|l|.mode = UNIT) or (|l|.mode = R-SPLIT)
9           wr := (wr ∪ |l|.reason) \ {l, l̄}
10        if(|l|.mode = L-SPLIT)
11          |l|.mode := R-SPLIT;
12          |l|.reason := wr;
13          return l̄;
14      else wr := wr \ {l};
15  return NULL;
```

Figure 3: A procedure for conflict-directed and solution-directed backjumping.

- given what we said in the paragraph below Theorem 3, the 2nd statement allows us to conclude that $\{\neg x_1, x_4\}$ is a reason for φ_μ satisfiability when μ is $\neg x_1; \neg x_3; x_4$,
- from the above, the 2nd statement allows us to conclude that $\{\neg x_1\}$ is a reason for φ_μ satisfiability when μ is $\neg x_1; \neg x_3$,
- from the above, the 1st statement allows us to conclude that $\{\neg x_1\}$ is a reason for φ_μ satisfiability when μ is $\neg x_1$.

From the last item, it follows that looking for assignments falsifying φ_μ when μ begins with $\neg x_1$, is useless. Given this, our "backjumping" procedure would have avoided the generation of the branch leading to $(F5)$ in Figure 2.

5 Implementation in QUBE

A procedure incorporating both conflict-directed and solution-directed backjumping has been implemented in QUBE. A high-level description of this procedure is presented in Figure 3. Consider Figure 3. Assume that φ is the input QBF and that $\mu = l_1; \ldots; l_m$ is the assignment for φ corresponding to the sequence of literals stored in the stack. Then,

- *InitWr*(*res*) initializes the variable wr, storing the reason for φ_μ result. In our implementation:
  - if *res* $=$ FALSE, φ_μ contains a contradictory clause. Let C be a clause in φ such that, for each literal l in C, $\bar{l}$ is in μ or l is universal in φ. Then *InitWr*(*res*) returns the set of literals l in μ such that $\bar{l}$ is in C (see Theorem 1). For example, if φ is (2), and μ is $x_1; \neg x_2; x_5; \neg x_3; x_4$, then *InitWr*(*res*) returns $\{x_1, \neg x_3, x_4\}$.
  - if *res* $=$ TRUE, the matrix of φ_μ is empty. Then *InitWr*(*res*) returns the set of literals in the sequence obtained from μ by recursively eliminating a universal literal l such that, for each clause C in φ, if $l \in C$ then there is another literal l' in the sequence with $l' \in C$ (see Theorem 3). For example, if φ is (2), and μ is $\neg x_1; \neg x_3; x_4; x_5$, then *InitWr*(*res*) returns $\{\neg x_1, x_4, x_5\}$.
- If l is a literal l_i in μ, $|l|.reason$ is the property of $|l|$ that, if set, stores the reason for $\varphi_{l_1;\ldots;l_i}$ result.

The function *QubeSolver* in Figure 1 needs to be modified in two ways. First, if l is a unit in φ_μ, then $\varphi_{\mu;\bar{l}}$ contains a contradictory clause. Let C be a clause in φ such that, for each literal l' in C, $\overline{l'}$ is in $\mu;\bar{l}$ or l' is universal in φ. Then, the set ν of literals l' in $\mu;\bar{l}$ such that $\overline{l'}$ is in C is a reason for $\varphi_{\mu;\bar{l}}$ unsatisfiability. If *UnitSetReason*(l) is invoked when l is a unit in φ_μ, and assuming this function returns a set ν defined as above, the instruction

$$|l|.reason := \textit{UnitSetReason}(l);$$

has to be added in line 8. For example, if φ is (2), and μ is $x_1; \neg x_2; x_5; \neg x_3$, then x_4 is a unit in φ_μ, $\varphi_{\mu;\neg x_4}$ contains a contradictory clause, and $\{x_1, \neg x_3, \neg x_4\}$ is stored in $x_4.reason$. Second, the procedure *Backjump*(*res*) has to be invoked in place of *Backtrack*(*res*) at line 25.

Considering the procedure *Backjump*(*res*) in Figure 3 — once the working reason is initialized (line 2) according to the above— *Backjump*(*res*) pops all the literals and corresponding QBFs (line 4) from the stack, till a literal l is reached such that l belongs to the working reason wr (line 5), $|l|.mode$ is L-SPLIT (line 10), and either

- l is existential and *res*=FALSE (line 6); or
- l is universal and *res*=TRUE (line 7).

If such a literal l exists, $|l|.mode$ is set to R-SPLIT(line 11), the working reason is stored in $|l|.reason$ (line 12), and $\bar{l}$ is returned (line 13). If no such literal exists, NULL is returned (line 15).

Notice that if l is not in wr, we can safely retract l despite the other conditions (see statement 1 in Theorems 2, 4): Assigning $\bar{l}$ would not change the result of the computation.

If l is in wr, one of the conditions in line 6 or line 7 is satisfied, but $|l|.mode$ is UNIT or R-SPLIT (line 8), we can use the results in statement 4 of Theorems 2, 4 to compute the new working reason (line 9).

If l is in wr but neither the condition in line 6 nor the condition in line 7 is satisfied, then we can retract l and remove l from wr (line 14). See the second statement of Theorems 2, 4.

Finally, given the reasons returned by our implementation of *InitWr*(*res*), it is easy to see that neither a universal, monotone literal can belong to the reason of an universal literal when *res*=TRUE, nor an existential, monotone literal can belong to the reason of an existential literal when *res*=FALSE. Thus, the statement 3 in Theorems 2, 4 (included for theoretical completeness) is never applied in our implementation.

Our implementation of QUBE with *Backjump* is a generalization of the conflict-directed backjumping procedure implemented, e.g., in RELSAT: assuming that our procedure and RELSAT –without learning– perform the same nondeterministic choices, the two systems have the same behavior on QBFs without universal quantifiers.

6 Experimental analysis

To evaluate the benefits deriving from backjumping, we compare QUBE with backtracking (that we call QUBE-BT), and QUBE with backjumping (that we call QUBE-BJ). For both

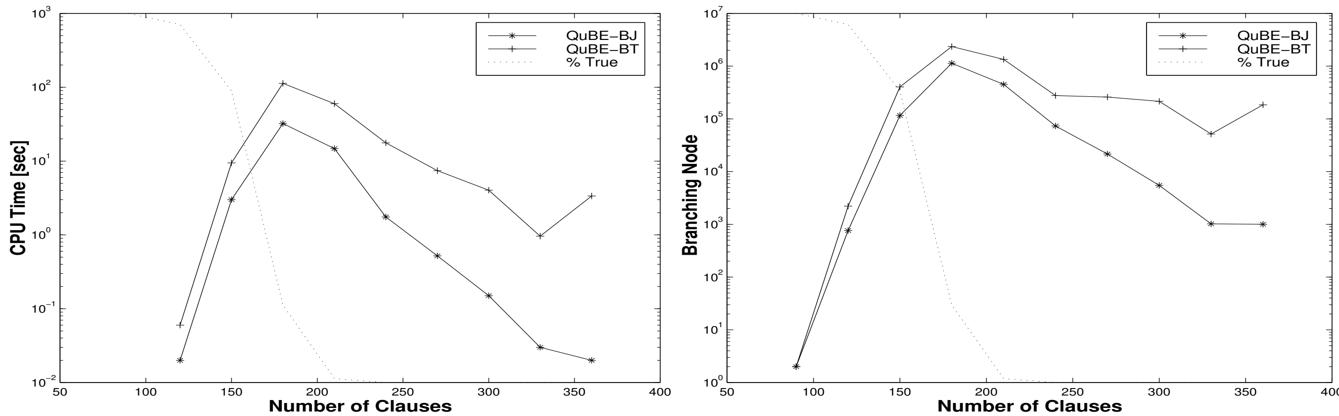

Figure 4: QUBE-BT and QUBE-BJ median CPU time (left) and number of branching nodes (right). 100 samples/point. Background: satisfiability percentage.

systems, we use the branching heuristics described in [Feldmann *et al.*, 2000]). All the tests have been run on a Pentium III, 600MHz, 128MBRAM.

We first consider sets of randomly generated QBFs. The generation model that we use is model A by Gent and Walsh [1999]. In this model, each QBF has the following 4 properties:

1. the prefix consists of k sequences, each sequence has n quantifiers, and each two quantifiers in a same sequence, are of the same type,
2. the rightmost quantifier is $\exists$,
3. the matrix consists of l clauses,
4. each clause consists of h literals of which at least 2 are existential.

Figure 4 shows the median of the CPU times (left) and number of branching nodes (right) of QUBE-BT and QUBE-BJ when $k = 4$, $n = 30$, $h = 5$, and l (on the x-axis) is varied in such a way to empirically cover the "100% satisfiable – 100% unsatisfiable" transition (shown in the background). Notice the logarithmic scale on the y-axis. Consider Figure 4-left. As it can be observed, QUBE-BJ is faster (up-to two orders of magnitude) than QUBE-BT. QUBE-BJ better performances are due to its minor number of branching nodes, as shown in Figure 4-right.

	QUBE-BT		QUBE-BJ	
Test File	Time	Br. Nodes	Time	Br. Nodes
B*-3ii.4.3	>1200	–	2.340	59390
B*-3ii.5.2	>1200	–	53.500	525490
B*-3iii.4	>1200	–	0.560	18952
T*-6.1.iv.11	31.990	500471	6.860	139505
T*-6.1.iv.12	22.700	298172	2.660	56387
T*-7.1.iv.13	755.760	8932409	117.380	1948541
T*-7.1.iv.14	466.600	4729695	38.720	688621
C*-23.24	604.020	8388653	706.430	8388653

Table 1: QUBE-BT and QUBE-BJ on Rintanen's benchmarks. Names have been abbreviated to fit in the table.

We also test QUBE-BT and QUBE-BJ on the structured Rintanen's benchmarks.[3] These problems are translations from planning problems to QBF (see [Rintanen, 1999a]). For lack of space, Table 1 shows QUBE-BT and QUBE-BJ performances on only 8 out of the 38 benchmarks available. On 13 of the other 30 problems, QUBE-BT and QUBE-BJ perform the same number of branching nodes, and on the remaining 17, none of the two systems is able to solve the problem in the time limit of 1200 seconds. As before, QUBE-BJ never performs more branching nodes than QUBE-BT, and is sometimes much faster. The last line of the table shows that when the number of branching nodes performed by QUBE-BT and QUBE-BJ is the same, the computational overhead paid by QUBE-BJ is not dramatic.

7 Conclusions and Future work

In the paper we have shown that it is possible to generalize the conflict-directed backjumping schema for SAT to QBFs. We have introduced solution-directed backjumping. Our implementation in QUBE shows that these forms of backjumping can produce significant speed ups. As far as we know, this is the first time a backjumping schema has been proposed, implemented and experimented for QBFs. It is worth remarking that the logics of conflict-directed (section 4.1) and solution-directed (section 4.2) backjumping are symmetrical: Indeed, it would have been easy to provide a uniform treatment accounting for both forms of backjumping. We decided not to do it in order to improve the readability of the paper, at the same time showing the tight relations with the conflict-directed backjumping schema for SAT.

Beside the above results, we have also comparatively tested QUBE and some of the other QBF solvers mentioned in the introduction. Our results show that QUBE compares well with respect to the other deciders, even without backjumping. For example, of the 38 Rintanen's structured problems, DECIDE, QUBE-BJ, QUBE-BT, QSOLVE, EVALUATE are able to solve 35, 21, 18, 11, 0 samples respectively in less than 1200s. In this regard, we point out that DECIDE features

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

"inversion of quantifiers" and "sampling" mechanisms which seem particularly effective on these benchmarks.

QuBE, the experimental results, and the test sets used are available at QuBE web page:

`www.mrg.dist.unige.it/star/qube.`

QuBE, besides the backjumping procedure above described, features six different branching heuristics, plus an adaptation of trivial truth (see [Cadoli *et al.*, 1998, Cadoli *et al.*, 2000]). (See [Giunchiglia *et al.*, 2001b] for a description of QuBE's available options.) About trivial truth, backjumping and their interactions, we have conducted an experimental evaluation on this. The result is that neither trivial truth is always better than backjumping, nor the other way around. On the other hand, the overhead of each of these techniques (on the test sets we have tried) is not dramatic, and thus it seems a good idea to use both of them. These and other results are reported in [Giunchiglia *et al.*, 2001c].

About the future work, we are currently investigating the effects of using different branching heuristics. Then, we plan to extend QuBE in order to do some form of "size" or "relevance" learning as it has been done in SAT (see, e.g., [Bayardo, Jr. and Schrag, 1997]).

Acknowledgments

Thanks to Davide Zambonin for the many fruitful discussion about implementation. Thanks to an anonymous reviewer for suggesting the name "solution-directed backjumping". Thanks to Marco Cadoli, Rainer Feldmann, Theodor Lettman, Jussi Rintanen, Marco Schaerf and Stefan Schamberger for providing us with their systems and helping us to figure them out during our experimental analisys. This work has been partially supported by ASI and MURST.

References

[Bayardo, Jr. and Schrag, 1997] R. J. Bayardo, Jr. and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. AAAI*, pages 203–208, 1997.

[Cadoli *et al.*, 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proc. AAAI*, 1998.

[Cadoli *et al.*, 2000] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 2000. To appear. Reprinted in [Gent *et al.*, 2000].

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, January 1990.

[Egly *et al.*, 2000] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In *Proc. AAAI*, 2000.

[Feldmann *et al.*, 2000] R. Feldmann, B. Monien, and S. Schamberger. A distributed algorithm to evaluate quantified boolean formulae. In *Proc. AAAI*, 2000.

[Gent and Walsh, 1999] Ian Gent and Toby Walsh. Beyond NP: the QSAT phase transition. In *Proc. AAAI*, pages 648–653, 1999.

[Gent *et al.*, 2000] Ian P. Gent, Hans Van Maaren, and Toby Walsh, editors. *SAT2000. Highlights of Satisfiability Research in the Year 2000*. IOS Press, 2000.

[Giunchiglia *et al.*, 2001a] Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001)*, 2001.

[Giunchiglia *et al.*, 2001b] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE: A system for deciding quantified boolean formulas satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001)*, 2001.

[Giunchiglia *et al.*, 2001c] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. On the effectiveness of backjumping and trivial truth in quantified boolean formulas satisfiability, 2001. Submitted. Available at `www.mrg.dist.unige.it/star/qube`.

[Kleine-Büning, H. and Karpinski, M. and Flögel, A., 1995] Kleine-Büning, H. and Karpinski, M. and Flögel, A. Resolution for quantified boolean formulas. *Information and computation*, 117(1):12–18, 1995.

[Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Rintanen, 1999a] Jussi Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rintanen, 1999b] Jussi T. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *Proc. IJCAI*, pages 1192–1197, 1999.