Incremental QBF Solving

Florian Lonsing and Uwe Egly

Institute of Information Systems
Vienna University of Technology
http://www.kr.tuwien.ac.at/
Vienna, Austria

20th International Conference on Principles and Practice of Constraint Programming, September 8 - 12, Lyon, France

This work is supported by the Austrian Science Fund (FWF) under grant S11409-N23.

Overview (1/3)

Quantified Boolean Formulas (QBF):

- Propositional formulae with universally ($\forall$) and ($\exists$) existentially quantified variables.
- In terms of QCSP: all variables have Boolean domains, all constraints are clauses. E.g. $\exists x \forall y \exists z.\ C_1 \wedge C_2 \wedge \ldots \wedge C_n$.
- Solving a QBF: PSPACE-complete.
- Applications in model checking, formal verification, testing,...

Incremental Solving:

- In practice, often a sequence $\psi_0, \psi_1, \ldots, \psi_n$ of related formulas must be solved.
- Try to exploit similarity between formulas in a sequence.
- Information gathered when solving ψ_i might help to solve ψ_j with $j > i$.

Overview (1/3)

Quantified Boolean Formulas (QBF):

- Propositional formulae with universally ($\forall$) and ($\exists$) existentially quantified variables.
- In terms of QCSP: all variables have Boolean domains, all constraints are clauses. E.g. $\exists x \forall y \exists z.\ C_1 \wedge C_2 \wedge \ldots \wedge C_n$.
- Solving a QBF: PSPACE-complete.
- Applications in model checking, formal verification, testing,...

Incremental Solving:

- In practice, often a sequence $\psi_0, \psi_1, \ldots, \psi_n$ of related formulas must be solved.
- Try to exploit similarity between formulas in a sequence.
- Information gathered when solving ψ_i might help to solve ψ_j with $j > i$.

Overview (2/3): Non-Incremental QBF Solving

$$\psi_0 \longrightarrow \boxed{\text{Solver}} \longrightarrow \text{SAT/UNSAT}$$

$$\psi_1 \longrightarrow \boxed{\text{Solver}} \longrightarrow \text{SAT/UNSAT}$$

$$\vdots$$

$$\psi_n \longrightarrow \boxed{\text{Solver}} \longrightarrow \text{SAT/UNSAT}$$

- Given: sequence $\psi_0, \psi_1, \ldots, \psi_n$ of PCNFs to be solved.
- Typical usage scenario: solver is called from the command line.
- Each ψ_i is parsed from scratch (might incur non-negligible overhead).
- Syntactic similarity between ψ_i and ψ_j with $i < j$ is not exploited.
- All information gathered when solving ψ_i is lost.
- Potential repetition of work when solving ψ_{i+1}.

Overview (3/3)

Incremental QBF Solving:

- Overview of general-purpose incremental QBF solving.
- Backtracking search procedure based on DPLL algorithm for QBF.
- Proof system: derivation of learned constraints by resolution.
- Challenge: which constraints can be reused in incremental solving?
- Promising experimental results.

DepQBF:

- Incremental QBF solver.
- Free software: `http://lonsing.github.io/depqbf/`
- Related work:
  - Incremental SAT solving by MiniSAT,... [ES03, NRS14].
  - Incremental QBF solving by QuBE: bounded model checking of partial designs [MMLB12].

Overview (3/3)

Incremental QBF Solving:

- Overview of general-purpose incremental QBF solving.
- Backtracking search procedure based on DPLL algorithm for QBF.
- Proof system: derivation of learned constraints by resolution.
- Challenge: which constraints can be reused in incremental solving?
- Promising experimental results.

DepQBF:

- Incremental QBF solver.
- Free software: `http://lonsing.github.io/depqbf/`
- Related work:
  - Incremental SAT solving by MiniSAT,... [ES03, NRS14].
  - Incremental QBF solving by QuBE: bounded model checking of partial designs [MMLB12].

QBF Syntax

QBF in Prenex Conjunctive Normal Form:

- Given a Boolean formula $\phi(x_1, \ldots, x_m)$ in CNF.
- Quantifier prefix $\hat{Q} := Q_1 B_1 Q_2 B_2 \ldots Q_m B_m$.
- Quantifiers $Q_i \in \{\forall, \exists\}$.
- Quantifier block $B_i \subseteq \{x_1, \ldots, x_m\}$ containing variables.
- QBF in prenex CNF (PCNF): $Q_1 B_1 Q_2 B_2 \ldots Q_m B_m.\phi(x_1, \ldots, x_m)$.
- $B_i \leq B_{i+1}$: quantifier blocks are linearly ordered (extended to variables, literals).

Example

- Given the CNF $\phi := (x \vee \neg y) \wedge (\neg x \vee y)$.
- Given the quantifier prefix $\hat{Q} := \forall x \exists y$.
- Prenex CNF: $\psi := \hat{Q}.\phi = \forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y)$.

QBF Semantics (1/2)

Recursive Definition:

- Given a PCNF $\psi := Q_1 B_1 \ldots Q_m B_m.\ \phi$.
- Prerequisite: every variable is quantified in the prefix (no free variables).
- Recursively assign the variables in prefix order (from left to right).
- Base cases: the QBF $\top$ ($\bot$) is satisfiable (unsatisfiable).
- $\psi = \forall x \ldots \phi$ is satisfiable if $\psi[\neg x]$ and $\psi[x]$ are satisfiable.
- $\psi = \exists x \ldots \phi$ is satisfiable if $\psi[\neg x]$ or $\psi[x]$ is satisfiable.
- In $\psi[x]$ ($\psi[\neg x]$), every occurrence of x in ψ is replaced by $\top$ ($\bot$).
- Assignment $A = \{l_1, \ldots, l_n\}$: if $l_i \in A$ is a positive (negative) literal, then $var(l_i)$ is assigned to true (false).

QBF Semantics (2/2)

Example (continued)

The PCNF $\psi = \forall x \exists y.(x \vee \neg y) \wedge (\neg x \vee y)$ is satisfiable if

(1) $\psi[x] = \exists y.(y)$ and

(2) $\psi[\neg x] = \exists y.(\neg y)$ are satisfiable.

(1) $\psi[x] = \exists y.(y)$ is satisfiable since $\psi[x, y] = \top$ is satisfiable.
(2) $\psi[\neg x] = \exists y.(\neg y)$ is satisfiable since $\psi[\neg x, \neg y] = \top$ is satisfiable.

Search-Based QBF Solving with Constraint Learning

- QBF-specific variant of DPLL algorithm [DLL62, CGS98].
- Basic idea: backtracking-based implementation of semantics by enumeration of assignments.
- *Constraint learning* as a proof system, based on enumerated assignments A.
- $\psi[A] = \bot$: derive a new *clause*.
- $\psi[A] = \top$: derive a new *cube*, i.e. conjunction of literals.
- Derivation relation $\vdash$.

Very high-level view, omitting crucial details:

```
bool bt_search (PCNF Qxψ, Assignment A)
   /* 1. Simplify under given assignment. */
       ψ' := simplify(Qxψ[A]);
   /* 2. Check base cases. */
       if (ψ' == ⊥)
         return false;
       if (ψ' == ⊤)
         return true;
   /* 3. Assignment generation, backtracking,
         constraint learning */
       if (Q == ∃)
         return bt_search (ψ', A ∪ {¬x}) ||
                bt_search (ψ', A ∪ {x});
       if (Q == ∀)
         return bt_search (ψ', A ∪ {¬x}) &&
                bt_search (ψ', A ∪ {x});
```

Constraint Learning by Example (1/2): Clause Derivations

Example

$\psi := \exists x \forall u \exists y.\ (x \vee u \vee \neg y) \wedge (x \vee u \vee y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge (\neg x \vee \neg u \vee y).$

$$(x \vee u \vee \neg y) \qquad (x \vee u \vee y) \qquad (\neg x \vee \neg u \vee \neg y) \qquad (\neg x \vee \neg u \vee y)$$

- Input clauses: $\forall C \in \psi$, by definition it holds that $\psi \vdash C$.

Constraint Learning by Example (1/2): Clause Derivations

Example

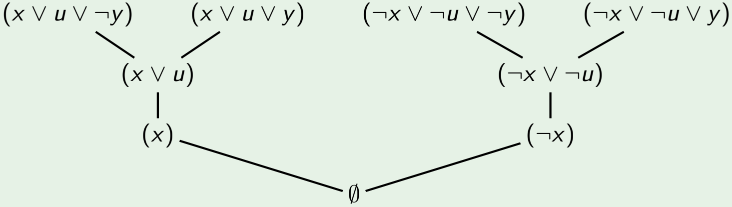
$\psi := \exists x \forall u \exists y.\ (x \vee u \vee \neg y) \wedge (x \vee u \vee y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge (\neg x \vee \neg u \vee y).$

$$\dfrac{(x \vee u \vee \neg y) \qquad (x \vee u \vee y)}{(x \vee u)} \qquad\qquad \dfrac{(\neg x \vee \neg u \vee \neg y) \qquad (\neg x \vee \neg u \vee y)}{(\neg x \vee \neg u)}$$

- Input clauses: $\forall C \in \psi$, by definition it holds that $\psi \vdash C$.
- Resolution of clauses (informally): for C_1, C_2 with $\psi \vdash C_1$, $\psi \vdash C_2$ and $x \in C_1$ and $\neg x \in C_2$, it holds that $\psi \vdash C$ where $C = C_1 \otimes C_2$ is the resolvent of C_1 and C_2.

Example

$\psi := \exists x \forall u \exists y.\ (x \vee u \vee \neg y) \wedge (x \vee u \vee y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge (\neg x \vee \neg u \vee y).$



- Input clauses: $\forall C \in \psi$, by definition it holds that $\psi \vdash C$.
- Resolution of clauses (informally): for C_1, C_2 with $\psi \vdash C_1$, $\psi \vdash C_2$ and $x \in C_1$ and $\neg x \in C_2$, it holds that $\psi \vdash C$ where $C = C_1 \otimes C_2$ is the resolvent of C_1 and C_2.
- Reduction of clauses: for C_1 with $\psi \vdash C_1$, it holds that $\psi \vdash C$ where C is obtained by removing trailing universal literals from C by prefix ordering.

Constraint Learning by Example (2/2): Cube Derivations

Example

$\psi := \forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y)$.

$$\begin{array}{ccc} \psi[x, y] = \top & \qquad & \psi[\neg x, \neg y] = \top \\ | & & | \\ (x \wedge y) & & (\neg x \wedge \neg y) \end{array}$$

- Model generation: for an assignment A with $\psi[A] = \top$, it holds that $\psi \vdash C$ where $C = \bigwedge_{l \in A} l$.

Constraint Learning by Example (2/2): Cube Derivations

Example

$\psi := \forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y)$.

$$
\begin{array}{ccc}
\psi[x, y] = \top & \qquad & \psi[\neg x, \neg y] = \top \\
| & & | \\
(x \wedge y) & & (\neg x \wedge \neg y) \\
| & & | \\
(x) & & (\neg x)
\end{array}
$$

- Model generation: for an assignment A with $\psi[A] = \top$, it holds that $\psi \vdash C$ where $C = \bigwedge_{l \in A} l$.
- Reduction of cubes: for C_1 with $\psi \vdash C_1$, it holds that $\psi \vdash C$ where C is obtained by removing trailing existential literals from C by prefix ordering.

Constraint Learning by Example (2/2): Cube Derivations

Example

$\psi := \forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y).$

$$
\begin{tikzcd}
\psi[x, y] = \top \arrow[d, dash] & & \psi[\neg x, \neg y] = \top \arrow[d, dash] \\
(x \wedge y) \arrow[d, dash] & & (\neg x \wedge \neg y) \arrow[d, dash] \\
(x) \arrow[dr, dash] & & (\neg x) \arrow[dl, dash] \\
& \emptyset &
\end{tikzcd}
$$

- Model generation: for an assignment A with $\psi[A] = \top$, it holds that $\psi \vdash C$ where $C = \bigwedge_{l \in A} l$.
- Reduction of cubes: for C_1 with $\psi \vdash C_1$, it holds that $\psi \vdash C$ where C is obtained by removing trailing existential literals from C by prefix ordering.
- Resolution of cubes (informally): for C_1, C_2 with $\psi \vdash C_1$, $\psi \vdash C_2$ and $x \in C_1$ and $\neg x \in C_2$, it holds that $\psi \vdash C$ where $C = C_1 \otimes C_2$ is the resolvent of C_1 and C_2.

Constraint Learning as a Proof System

Satisfiability:

- Soundness of a learned cube C with $\psi \vdash C$: $\hat{Q}.\phi \equiv_{sat} \hat{Q}.(\phi \vee C)$.
- Derivation of empty cube: $\psi \vdash \emptyset$ if and only if ψ satisfiable.

Unsatisfiability:

- Soundness of a learned clause C with $\psi \vdash C$: $\hat{Q}.\phi \equiv_{sat} \hat{Q}.(\phi \wedge C)$.
- Derivation of empty clause: $\psi \vdash \emptyset$ if and only if ψ unsatisfiable.

Clause and Cube Learning in Search-Based QBF Solving:

- Assignment generation drives the application of the proof rules.

Clause and Cube Learning in Incremental Solving:

- If ψ is modified to obtain ψ', then if $\psi \vdash C$ for a constraint C we might have $\psi' \nvdash C$.
- E.g.: if $\psi' \nvdash C$ for a clause C then in general $\psi' \not\equiv_{sat} \psi' \wedge C$.
- Soundness: non-derivable (potentially invalid) constraints must be discarded.

Constraint Learning as a Proof System

Satisfiability:

- Soundness of a learned cube C with $\psi \vdash C$: $\hat{Q}.\phi \equiv_{sat} \hat{Q}.(\phi \vee C)$.
- Derivation of empty cube: $\psi \vdash \emptyset$ if and only if ψ satisfiable.

Unsatisfiability:

- Soundness of a learned clause C with $\psi \vdash C$: $\hat{Q}.\phi \equiv_{sat} \hat{Q}.(\phi \wedge C)$.
- Derivation of empty clause: $\psi \vdash \emptyset$ if and only if ψ unsatisfiable.

Clause and Cube Learning in Search-Based QBF Solving:

- Assignment generation drives the application of the proof rules.

Clause and Cube Learning in Incremental Solving:

- If ψ is modified to obtain ψ', then if $\psi \vdash C$ for a constraint C we might have $\psi' \nvdash C$.
- E.g.: if $\psi' \nvdash C$ for a clause C then in general $\psi' \not\equiv_{sat} \psi' \wedge C$.
- Soundness: non-derivable (potentially invalid) constraints must be discarded.

Incremental Solving

$$
\begin{array}{lcccl}
\psi_0 & \longrightarrow & \boxed{\text{Solver}} & \longrightarrow & \text{SAT/UNSAT} \\
 & & \downarrow LC'_0 & & \\
\psi_1: & \xrightarrow{\phi_1^{del}, \phi_1^{add}} & \boxed{\text{Solver}} & \longrightarrow & \text{SAT/UNSAT} \\
 & & \vdots & & \\
 & & \downarrow LC'_{n-1} & & \\
\psi_n: & \xrightarrow{\phi_n^{del}, \phi_n^{add}} & \boxed{\text{Solver}} & \longrightarrow & \text{SAT/UNSAT}
\end{array}
$$

- Typical usage scenario: solver is called as a library from an external program via API.
- Reduced hard disk I/O overhead: only new parts are parsed.
- LC'_i: subset of the constraints learned when solving ψ_j with $j \leq i$.
- Parts of the constraints learned when solving previous formulas can be reused.
- Reused clause (cube) C: $\psi_{i+1} \equiv_{sat} \psi_{i+1} \wedge C$ ($\psi_{i+1} \equiv_{sat} \psi_{i+1} \vee C$) must still hold.
- Potential speed up compared to non-incremental solving.

Incremental Solving: Deleting Clauses from the Input Formula (1/2)

Example (continued)

$\psi := \exists x \forall u \exists y.\ (x \vee u \vee \neg y) \wedge (x \vee u \vee y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge (\neg x \vee \neg u \vee y).$

$(x \vee u \vee \neg y) \quad (x \vee u \vee y) \qquad (\neg x \vee \neg u \vee \neg y) \quad (\neg x \vee \neg u \vee y)$

$(x \vee u) \qquad\qquad (\neg x \vee \neg u)$

$(x) \qquad\qquad (\neg x)$

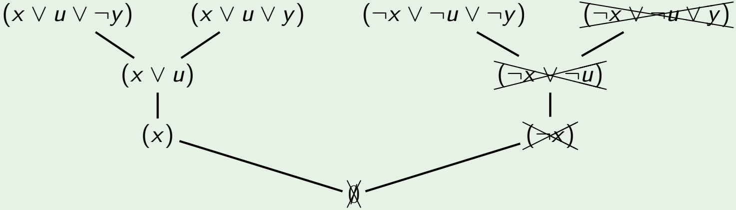
$\emptyset$

- *Deleting clauses* from ψ_i to obtain ψ_{i+1}: for a *learned clause* C with $\psi_i \vdash C$ we might have $\psi_{i+1} \nvdash C$ and $\psi_{i+1} \not\equiv_{sat} \psi_{i+1} \wedge C$.
- From ψ_i to ψ_{i+1}: the set of learned clauses must be maintained.
- How to detect efficiently if $\psi_{i+1} \vdash C$?
- In practice: solvers do not keep the derivations of the learned constraints.

Incremental Solving: Deleting Clauses from the Input Formula (1/2)

Example (continued)

$\psi := \exists x \forall u \exists y.\ (x \vee u \vee \neg y) \wedge (x \vee u \vee y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge \cancel{(\neg x \vee \neg u \vee y)}.$

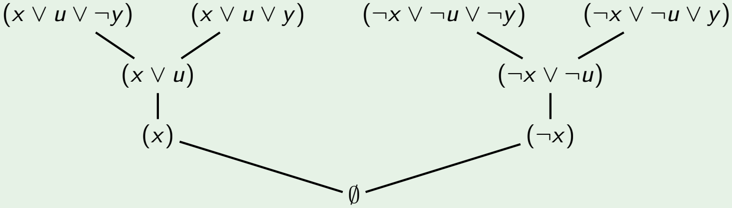


- *Deleting clauses* from ψ_i to obtain ψ_{i+1}: for a *learned clause* C with $\psi_i \vdash C$ we might have $\psi_{i+1} \nvdash C$ and $\psi_{i+1} \not\equiv_{sat} \psi_{i+1} \wedge C$.
- From ψ_i to ψ_{i+1}: the set of learned clauses must be maintained.
- How to detect efficiently if $\psi_{i+1} \vdash C$?
- In practice: solvers do not keep the derivations of the learned constraints.

Incremental Solving: Deleting Clauses from the Input Formula (2/2)

Example (continued)

$\psi := \exists x \forall u \exists y.$
$(x \vee u \vee \neg y) \wedge (x \vee u \vee y) \wedge (\neg x \vee \neg u \vee \neg y) \wedge (\neg x \vee \neg u \vee y).$



Incremental Solving: Deleting Clauses from the Input Formula (2/2)

Example (continued)

$\psi := \exists s_1, s_2, s_3, s_4, x \forall u \exists y.$
$(s_1 \lor x \lor u \lor \neg y) \land (s_2 \lor x \lor u \lor y) \land (s_3 \lor \neg x \lor \neg u \lor \neg y) \land (s_4 \lor \neg x \lor \neg u \lor y).$

$$(s_1 \lor x \lor u \lor \neg y) \quad (s_2 \lor x \lor u \lor y) \quad (s_3 \lor \neg x \lor \neg u \lor \neg y) \quad (s_4 \lor \neg x \lor \neg u \lor y)$$

$$(s_1 \lor s_2 \lor x \lor u) \qquad (s_3 \lor s_4 \lor \neg x \lor \neg u)$$

$$(s_1 \lor s_2 \lor x) \qquad (s_3 \lor s_4 \lor \neg x)$$

$$(s_1 \lor s_2 \lor s_3 \lor s_4)$$

- Selector variables: fresh, leftmost existential variables added to each input clause.
- Solving under predefined assignments to selector variables (called *assumptions*).
- Setting selector variables to false (true): clauses are enabled (disabled).
- "Empty clause" contains only selector variables, all of which are assigned false.

Example (continued)

$\psi := \exists s_1, s_2, s_3, s_4, x \forall u \exists y.$
$(s_1 \vee x \vee u \vee \neg y) \wedge (s_2 \vee x \vee u \vee y) \wedge (s_3 \vee \neg x \vee \neg u \vee \neg y) \wedge (\top \vee \neg x \vee \neg u \vee y).$

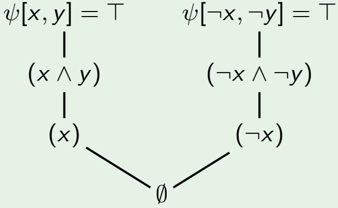
$$
\begin{tikzcd}[column sep=tiny]
(s_1 \vee x \vee u \vee \neg y) \arrow[dr, dash] & & (s_2 \vee x \vee u \vee y) \arrow[dl, dash] & & (s_3 \vee \neg x \vee \neg u \vee \neg y) \arrow[dr, dash] & & (\top \vee \neg x \vee \neg u \vee y) \arrow[dl, dash] \\
& (s_1 \vee s_2 \vee x \vee u) \arrow[d, dash] & & & & (s_3 \vee \top \vee \neg x \vee \neg u) \arrow[d, dash] & \\
& (s_1 \vee s_2 \vee x) \arrow[drr, dash] & & & & (s_3 \vee \top \vee \neg x) \arrow[dll, dash] & \\
& & & (s_1 \vee s_2 \vee s_3 \vee \top) & & &
\end{tikzcd}
$$

- Setting selector variables to true disables (effectively deletes) input clauses...
- ...and also depending derived clauses.
- Selector variables are common in incremental SAT solving.

Incremental Solving: Adding Clauses to the Input Formula

Example (continued)

$\psi := \forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y)$.



- *Adding clauses* to ψ_i to obtain ψ_{i+1}: for a *learned cube* C with $\psi_i \vdash C$ we might have $\psi_{i+1} \nvdash C$ and $\psi_{i+1} \not\equiv_{sat} \psi_{i+1} \vee C$.
- From ψ_i to ψ_{i+1}: the set of learned cubes must be maintained.
- Problem: assignments in model generation rule might no longer be satisfying.
- Selector variables not directly applicable (initial cubes are derived on-the-fly).
- In DepQBF: only derivable *initial* cubes are kept.

Incremental Solving: Adding Clauses to the Input Formula

Example (continued)

$\psi := \forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee y)$.

$$
\begin{array}{ccc}
\psi[x, y] = \top & & \psi[\neg x, \neg y] = \bot \\
| & & | \\
(x \wedge y) & & \cancel{(\neg x \wedge \neg y)} \\
| & & | \\
(x) & & \cancel{(\neg x)} \\
& \diagdown \quad \diagup & \\
& \cancel{\emptyset} &
\end{array}
$$

Adding the clause $(x \vee y)$ produces an unsatisfiable formula.

- *Adding clauses* to ψ_i to obtain ψ_{i+1}: for a *learned cube* C with $\psi_i \vdash C$ we might have $\psi_{i+1} \nvdash C$ and $\psi_{i+1} \not\equiv_{sat} \psi_{i+1} \vee C$.
- From ψ_i to ψ_{i+1}: the set of learned cubes must be maintained.
- Problem: assignments in model generation rule might no longer be satisfying.
- Selector variables not directly applicable (initial cubes are derived on-the-fly).
- In DepQBF: only derivable *initial* cubes are kept.

Experiments

QBFEVAL'12-SR-Bloqqer			
	discard LC	*keep LC*	*diff.(%)*
$\overline{a}$:	39.75×10^6	34.03×10^6	-14.40
$\tilde{a}$:	1.71×10^6	1.65×10^6	-3.62
$\overline{b}$:	117,019	91,737	-21.61
$\tilde{b}$:	10,322	8,959	-13.19
$\overline{t}$:	100.15	95.36	-4.64
$\tilde{t}$:	4.18	2.83	-32.29

QBFEVAL'12-SR-Bloqqer			
	discard LC	*keep LC*	*diff.(%)*
$\overline{a}$:	5.88×10^6	1.29×10^6	-77.94
$\tilde{a}$:	103,330	8,199	-92.06
$\overline{b}$:	31,489	3,350	-89.37
$\tilde{b}$:	827	5	-99.39
$\overline{t}$:	30.29	9.78	-67.40
$\tilde{t}$:	0.50	0.12	-76.00

Average and median number of assignments ($\overline{a}$ and $\tilde{a}$, respectively), backtracks ($\overline{b}$, $\tilde{b}$), and wall clock time ($\overline{t}$, $\tilde{t}$) in seconds on fully solved *sequences* of PCNFs.

Left:

- Solving *sequences* $S = \psi_0, \ldots, \psi_{10}$ of PCNFs, where clauses are only added to ψ_i to obtain ψ_{i+1}.
- Learned constraints are discarded (*discard LC*) and correct ones are kept (*keep LC*).

Right:

- Solving the reversed sequences $S' = \psi_9, \ldots, \psi_0$ of PCNFs after the original sequence $S = \psi_0, \ldots, \psi_9, \psi_{10}$ has been solved, where clauses are only deleted from ψ_{i+1} to obtain ψ_i.

Experiments

QBFEVAL'12-SR-Bloqqer			
	discard LC	*keep LC*	*diff.(%)*
$\overline{a}$:	39.75×10^6	34.03×10^6	-14.40
$\tilde{a}$:	1.71×10^6	1.65×10^6	-3.62
$\overline{b}$:	117,019	91,737	-21.61
$\tilde{b}$:	10,322	8,959	-13.19
$\overline{t}$:	100.15	95.36	-4.64
$\tilde{t}$:	4.18	2.83	-32.29

QBFEVAL'12-SR-Bloqqer			
	discard LC	*keep LC*	*diff.(%)*
$\overline{a}$:	5.88×10^6	1.29×10^6	-77.94
$\tilde{a}$:	103,330	8,199	-92.06
$\overline{b}$:	31,489	3,350	-89.37
$\tilde{b}$:	827	5	-99.39
$\overline{t}$:	30.29	9.78	-67.40
$\tilde{t}$:	0.50	0.12	-76.00

Average and median number of assignments ($\overline{a}$ and $\tilde{a}$, respectively), backtracks ($\overline{b}$, $\tilde{b}$), and wall clock time ($\overline{t}$, $\tilde{t}$) in seconds on fully solved *sequences* of PCNFs.

Left:

- Solving *sequences* $S = \psi_0, \ldots, \psi_{10}$ of PCNFs, where clauses are only added to ψ_i to obtain ψ_{i+1}.
- Learned constraints are discarded (*discard LC*) and correct ones are kept (*keep LC*).

Right:

- Solving the reversed sequences $S' = \psi_9, \ldots, \psi_0$ of PCNFs after the original sequence $S = \psi_0, \ldots, \psi_9, \psi_{10}$ has been solved, where clauses are only deleted from ψ_{i+1} to obtain ψ_i.

Conclusions

Incremental QBF Solving:

- Useful for solving sequences of related formulas.
- Benefits from similarity between formulas.
- Tight integration into tool frameworks: library API, reduced I/O overhead.
- Challenge: keeping learned constraints.
- Further incremental QBF applications and case studies needed.

DepQBF:

- Open source incremental QBF solver implemented in C.
- API to add sets of clauses in a stack-based way (push/pop).
- Related papers:
  - AISC 2014 (accepted): case study of conformant planning by incremental QBF solving.
  - ICMS 2014: API example, further experiments [LE14].

DepQBF Source Code: `http://lonsing.github.io/depqbf/`

Conclusions

Incremental QBF Solving:

- Useful for solving sequences of related formulas.
- Benefits from similarity between formulas.
- Tight integration into tool frameworks: library API, reduced I/O overhead.
- Challenge: keeping learned constraints.
- Further incremental QBF applications and case studies needed.

DepQBF:

- Open source incremental QBF solver implemented in C.
- API to add sets of clauses in a stack-based way (push/pop).
- Related papers:
  - AISC 2014 (accepted): case study of conformant planning by incremental QBF solving.
  - ICMS 2014: API example, further experiments [LE14].

DepQBF Source Code: `http://lonsing.github.io/depqbf/`

M. Cadoli, A. Giovanardi, and M. Schaerf.
An Algorithm to Evaluate Quantified Boolean Formulae.
In *AAAI/IAAI*, pages 262–267, 1998.

M. Davis, G. Logemann, and D. Loveland.
A Machine Program for Theorem-proving.
Commun. ACM, 5(7):394–397, 1962.

Niklas Eén and Niklas Sörensson.
Temporal Induction by Incremental SAT Solving.
Electr. Notes Theor. Comput. Sci., 89(4):543–560, 2003.

Florian Lonsing and Uwe Egly.
Incremental QBF Solving by DepQBF.
In Hoon Hong and Chee Yap, editors, *ICMS*, volume 8592 of *Lecture Notes in Computer Science*, pages 307–314. Springer, 2014.

Paolo Marin, Christian Miller, Matthew D. T. Lewis, and Bernd Becker.
Verification of Partial Designs using Incremental QBF Solving.
In Wolfgang Rosenstiel and Lothar Thiele, editors, *DATE*, pages 623–628. IEEE, 2012.

Alexander Nadel, Vadim Ryvchin, and Ofer Strichman.
Ultimately incremental sat.
In Carsten Sinz and Uwe Egly, editors, *SAT*, volume 8561 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2014.