Enhancing Search-Based QBF Solving by Dynamic Blocked Clause Elimination

Florian Lonsing[1](✉), Fahiem Bacchus[2], Armin Biere[3], Uwe Egly[1], and Martina Seidl[3]

[1] Knowledge-Based Systems Group, Vienna University of Technology, Vienna, Austria
florian.lonsing@tuwien.ac.at
[2] Department of Computer Science, University of Toronto, Toronto, Canada
[3] Institute for Formal Models and Verification, JKU Linz, Linz, Austria

Abstract. Among preprocessing techniques for quantified Boolean formula (QBF) solving, quantified blocked clause elimination (QBCE) has been found to be extremely effective. We investigate the power of dynamically applying QBCE in search-based QBF solving with clause and cube learning (QCDCL). This dynamic application of QBCE is in sharp contrast to its typical use as a mere preprocessing technique. In our dynamic approach, QBCE is applied eagerly to the formula interpreted under the assignments that have been enumerated in QCDCL. The tight integration of QBCE in QCDCL results in a variant of cube learning which is exponentially stronger than the traditional method. We implemented our approach in the QBF solver DepQBF and ran experiments on instances from the QBF Gallery 2014. On application benchmarks, QCDCL with dynamic QBCE substantially outperforms traditional QCDCL. Moreover, our approach is compatible with incremental solving and can be combined with preprocessing techniques other than QBCE.

1 Introduction

Quantified Boolean formulas (QBF) extend propositional logic with universal and existential quantifiers over propositional variables. QBFs potentially allow for exponentially more succinct encodings compared to plain propositional logic and provide a natural representation of applications which can be seen as two-player games [22] as found, e.g., in program synthesis or formal verification [2].

A typical representation of QBFs is *prenex conjunctive normal form (PCNF)*, consisting of a quantifier prefix and a propositional CNF. In the game-based QBF semantics a universal and an existential player play against each other. The players assign truth values to the respective variables in the order enforced by the quantifier prefix. The universal player aims to falsify the formula while the existential player aims to satisfy it. PCNF allows to detect more easily when

Supported by the Austrian Science Fund (FWF) under grants S11408-N23 and S11409-N23.

the universal player has won. Once the literals of one clause are all set to false, a conflict is detected and the universal player wins the current round of the game. In contrast, all clauses must be satisfied before we can detect a win for the existential player. This bias is reflected in the sizes of clauses (small) and cubes (large) derived by search-based QBF solvers with clause and cube learning (QCDCL) [8,16,28].

While the uniformity of a CNF representation simplifies reasoning [6], it also blurs information essential for solving [1]. As a remedy, structural solvers directly operate on non-CNF formulas [5,9,15]. Dual propagation approaches combine structural and CNF-based reasoning and consider both a CNF and a DNF representation at the same time. This can be done explicitly so that a DNF and a CNF encoding of a problem are solved independently in parallel [25] or it can be directly integrated in a solver [10,15,27]. However, these approaches require a structural formulation of the problem that is often not available.

Another option is preprocessing, e.g., [3,7,11,21,26] which uses alternative techniques to recover structural information from a CNF ϕ. The goal is to rewrite ϕ into a new CNF that is easier to solve. For most QBF solvers preprocessing is vital and has been integrated in most QBF-based solving tool chains. Modern SAT solvers go even further and interleave preprocessing and standard search. In this approach called *inprocessing* [13], preprocessing is applied in bounded fashion to the formula simplified by unit clauses derived during the search. Until this work inprocessing had not found its way into modern QBF solvers.

The QBF solver StruQS heuristically combines search-based solving and variable elimination [17]. The latter is a complete decision procedure potentially exponential in space. Its bounded variant is a powerful preprocessing technique [11]. Preprocessing, however, can only recover structural information before search. Structural solvers, on the other hand, can exploit such information during search. Hence in this work we investigate a tighter integration of *blocked clause elimination (QBCE)* [11] in QCDCL-based QBF solvers. While QBCE originally is a preprocessing technique applied to QBFs in CNF, here we apply it *dynamically* during the search process of QCDCL. This way, we leverage the power of QBCE as a technique to simulate structural reasoning on CNFs in QCDCL. In our dynamic approach, QBCE is applied to the CNF interpreted under the assignment that has currently been enumerated in QCDCL. We show that this tight integration of QBCE in QCDCL results in cube learning that is exponentially stronger than its traditional variant. Dynamic QBCE also influences clause learning in that clauses identified as redundant by QBCE are not used to produce learned clauses in the current search context. In addition to dynamic QBCE we also investigate inprocessing in a QBF solver using QBCE.

We implemented inprocessing based on QBCE and dynamic QBCE in the QCDCL-based QBF solver DepQBF. On application benchmarks from the QBF Gallery 2014, QCDCL with dynamic QBCE *substantially* outperforms traditional QCDCL in terms of solved instances, run time and backtracks. We also observed a performance gain with inprocessing. We report on the details of our implementation of dynamic QBCE. Since QBCE is applied frequently during the search in QCDCL, sophisticated data structures are necessary to limit the computational costs. Dynamic QBCE is compatible with incremental solving and is extensible in that it can be combined with preprocessing techniques other than QBCE.

2 Preliminaries

We consider QBFs in *prenex conjunctive normal form* (PCNF). A QBF $\Pi.\psi$ in PCNF consists of a prefix Π and a matrix ψ. The prefix Π has the form $Q_1X_1Q_2X_2\dots Q_nX_n$ with disjoint variable sets X_i and $Q_i \in \{\forall,\exists\}$. Furthermore, $Q_i \neq Q_{i+1}$ and $\mathsf{var}(\Pi) = X_1 \cup \ldots \cup X_n$. We consider only closed QBFs: the matrix ψ of a QBF $\Pi.\psi$ contains only variables that occur in Π. The matrix ψ is a propositional formula in conjunctive normal form, i.e., a conjunction of clauses. A *clause* (*cube*) is a disjunction (conjunction) of literals. A *literal* is either a variable x or a negated variable $\bar{x}$. The negation of a literal l is denoted by $\bar{l}$. If convenient, we consider clauses and cubes as sets of literals. The variable of a literal is denoted by $\mathsf{var}(l)$ where $\mathsf{var}(l) = x$ if $l = x$ or $l = \bar{x}$. The quantifier $\mathsf{Q}(\Pi, l)$ of a literal l is Q_i if $\mathsf{var}(l) \in X_i$. Let $\mathsf{Q}(\Pi, l) = Q_i$ and $\mathsf{Q}(\Pi, k) = Q_j$, then $l \leq_\Pi k$ iff $i \leq j$.

A set of literals $A = \{l_1, \dots, l_n\}$ is called *assignment* of the QBF $\Pi.\psi$ if $\{\mathsf{var}(l_i) \mid l_i \in A\} \subseteq \mathsf{var}(\Pi)$ and for any $l_i, l_j \in A$ with $l_i \neq l_j$, $\mathsf{var}(l_i) \neq \mathsf{var}(l_j)$. By $\phi[A]$ we denote the QBF ϕ *under assignment* A, i.e., for $l \in A$, all clauses containing l are removed, all occurrences of $\bar{l}$ are deleted, and $\mathsf{var}(l)$ is removed from the prefix. The empty matrix is satisfiable, the matrix containing the empty clause is unsatisfiable. If the matrix of $\phi[A]$ is empty, then A is a *satisfying assignment* (written as $\phi[A] = \mathsf{T}$). If the matrix of $\phi[A]$ contains the empty clause, then A is a *falsifying assignment* (written as $\phi[A] = \mathsf{F}$). For a cube $C = (l_1 \wedge \ldots \wedge l_n)$, the set $\{l_1, \dots, l_n\}$ is the assignment defined by C. We write $\phi[C]$ to denote ϕ under the assignment defined by C. A closed QBF $\Pi.\psi$ with $Q_1 = \exists$ (resp. $Q_1 = \forall$) is satisfiable iff $\Pi.\psi[\{x\}]$ or (resp. and) $\Pi.\psi[\{\bar{x}\}]$ is satisfiable where $x \in X_1$. Two PCNFs ϕ and ϕ' are *satisfiability-equivalent*, written as $\phi \equiv_{sat} \phi'$, if and only if ϕ is satisfiable whenever ϕ' is satisfiable.

We introduce the *Q-resolution calculus* as a proof system which underlies search-based QBF solving with clause and cube learning [8,14,16,28].

Definition 1 (Q-Resolution Calculus). *Let $\phi = \Pi.\psi$ be a PCNF. The rules of the* Q-resolution calculus (QRES) *are as follows.*

$$\frac{C_1 \cup \{p\} \qquad C_2 \cup \{\bar{p}\}}{C_1 \cup C_2} \quad \begin{array}{l} \text{if } \{x,\bar{x}\} \not\subseteq (C_1 \cup C_2),\ \bar{p} \notin C_1,\ p \notin C_2 \\ \text{and either} \\ \text{(1) } C_1, C_2 \text{ are clauses and } \mathsf{Q}(\Pi,p) = \exists \text{ or} \\ \text{(2) } C_1, C_2 \text{ are cubes and } \mathsf{Q}(\Pi,p) = \forall \end{array} \qquad (res)$$

$$\frac{C \cup \{l\}}{C} \quad \begin{array}{l} \text{if } \{x,\bar{x}\} \not\subseteq (C \cup \{l\}) \text{ and either} \\ \text{(1) } C \text{ is a clause, } \mathsf{Q}(\Pi,l) = \forall, \\ \qquad l' <_\Pi l \text{ for all } l' \in C \text{ with } \mathsf{Q}(\Pi,l') = \exists \text{ or} \\ \text{(2) } C \text{ is a cube, } \mathsf{Q}(\Pi,l) = \exists, \\ \qquad l' <_\Pi l \text{ for all } l' \in C \text{ with } \mathsf{Q}(\Pi,l') = \forall \end{array} \qquad (red)$$

$$\frac{}{\text{C}} \quad \begin{array}{l}\text{if } \{x, \bar{x}\} \not\subseteq C \text{ and either} \\ (1)\ C \text{ is a clause and } C \in \psi \text{ or} \\ (2)\ C \text{ is a cube and } \phi[C] = \emptyset\end{array} \qquad (init)$$

Note that $\phi[C] = \emptyset$ in case (2) of rule *init* means that the matrix of $\phi[C]$ is empty. We write $\Pi.\psi \vdash C$ to denote that a clause or cube C is derivable from the PCNF $\Pi.\psi$ by rules *init*, *red*, and *res*. In a derivation of a clause (cube), the rules *init*, *red*, and *res* operate only on clauses (cubes). Q-resolution of clauses [14] is a generalization of propositional resolution, which is given by rules *init* and *res* when applied to clauses. Q-resolution of cubes was introduced in the context of solving satisfiable PCNFs [8,16,28]. Applications of rule *red* to clauses (cubes) are called *universal (existential) reduction*. We write $UR(C)$ ($ER(C)$) to denote the clause (cube) resulting from universal (existential) reduction of C. The PCNF $UR(\phi)$ is obtained by universal reduction of all clauses in the PCNF ϕ.

QRES is sound and refutationally complete for PCNFs [8,14,16,28]. The empty clause (cube) $C = \emptyset$ is derivable from a PCNF ϕ if and only if ϕ is unsatisfiable (satisfiable). A derivation of the empty clause (cube) from ϕ is a *clause (cube) resolution proof* of ϕ. The clausal variant of rule *res* is the basis for the definition of blocked clauses which is as follows.

Definition 2 (Blocked Clause). *A literal l with $\mathsf{Q}(\Pi, l) = \exists$ in a clause $C \in \psi$ of a QBF $\phi = \Pi.\psi$ is a* blocking literal *if for all $C' \in \psi$ with $\bar{l} \in C'$, a literal l' with $l' \leq_\Pi l$ exists such that $l', \bar{l'} \in C \cup (C' \setminus \{\bar{l}\})$. A clause is* blocked *if it contains a blocking literal.*

Note that blocking literal l in Definition 2 must be existential whereas literals $l', \bar{l'}$ can be existential or universal. *Blocked clause elimination (QBCE)* [11] removes blocked clauses from a PCNF ϕ until completion and takes time polynomial in the size of ϕ. The resulting PCNF is satisfiability-equivalent to ϕ.

3 Search-Based QBF Solving with Learning

In order to present our approach of dynamically applying QBCE in search-based QBF solvers, which results in a powerful variant of cube learning, we review the basic concepts of search-based QBF solving with learning.

Search-based QBF solving is based on a QBF-specific variant of the DPLL algorithm [4]. Similar to conflict-driven clause learning (CDCL) in SAT solving [23], search-based QBF solving has been equipped with clause and cube learning [8,16,28], called QCDCL. We briefly describe QCDCL based on the pseudo code shown in Fig. 1.

In QCDCL, assignments to the variables in a given input PCNF $\phi = \Pi.\psi$ are successively generated. Initially, the current assignment A is empty. During a run, ϕ is interpreted under A and $\phi[A]$ is simplified in a QBF-specific variant of *Boolean constraint propagation (QBCP)* in function `qbcp`. Additionally, in QBCP universal reduction according to rule *red* is applied to all clauses $C \in \phi[A]$, resulting in the PCNF $UR(\phi[A])$ with potentially shortened clauses $UR(C) \subseteq C$.

```
Result qcdcl (PCNF $\phi$)
  Result R = UNDEF;
  Assignment A = $\emptyset$;
  while (true)
    /* Simplify under A. */
    (R,A) = qbcp($\phi$,A);
    if (R == UNDET)
      /* Decision making. */
      A = assign_dec_var($\phi$,A);
    else
      /* Backtracking. */
      /* R == UNSAT/SAT */
      B = analyze(R,A);
      if (B == INVALID)
        return R;
      else
        A = backtrack(B);
```

Fig. 1. Pseudo code of QCDCL.

Assignment A is extended based on the detection of *unit* and *pure literals* in $UR(\phi[A])$. A clause $UR(C[A]) = (l)$, with $UR(C[A]) \in UR(\phi[A])$ containing the single literal l and $\mathsf{Q}(\Pi, l) = \exists$, is *unit* in $UR(\phi[A])$. Given such a unit clause, A is extended to $A := A \cup \{l\}$ and C is recorded as the *antecedent clause* of the assignment $\{l\}$. A literal l is *pure* in $UR(\phi[A])$ if $\bar{l}$ does not occur in $UR(\phi[A])$. Given a pure literal l, A is extended to $A := A \cup \{l\}$ if $\mathsf{Q}(\Pi, l) = \exists$ and to $A := A \cup \{\bar{l}\}$ if $\mathsf{Q}(\Pi, l) = \forall$. Simplifications of $\phi[A]$ and unit and pure literal detection are always applied until completion in QBCP. Assignments l_i in $A = \{l_1, \ldots, l_n\}$ are ordered chronologically.

If $\phi[A] \neq \mathsf{F}$ and $\phi[A] \neq \mathsf{T}$, then the satisfiability of $\phi[A]$ is still undetermined (`R == UNDET`). Some variable from the leftmost quantifier block of $\phi[A]$ is selected and tentatively assigned a value, thus extending A. Making tentative assignments is also called *decision making.*

If $\phi[A] = \mathsf{F}$ then $\phi[A]$ contains a clause C so that $UR(C[A]) = \emptyset$ is empty. If $\phi[A] = \mathsf{T}$, then $\phi[A]$ reduces to the empty matrix under A. In either case, the satisfiability of $\phi[A]$ has been determined (`R == UNSAT` or `R == SAT`). Assignment A is analyzed based on whether $\phi[A] = \mathsf{F}$ or $\phi[A] = \mathsf{T}$. A subset $B \subseteq A$ of assignment A is identified and retracted during backtracking. The run proceeds with the new, current assignment A obtained by backtracking. QCDCL generates assignments which have the following properties.

Definition 3 (QCDCL Assignment). *Given a QBF $\phi = \Pi.\psi$. Let assignment $A = A' \cup A''$ where A' are variables assigned in decision making and A'' are variables assigned by unit/pure literal detection. A is a* QCDCL assignment *if (1) for a maximal $l \in A'$ with $\forall l' \in A' : l' \leq_\Pi l$ it holds that $\forall x <_\Pi l : x \in \mathsf{var}(A)$ and (2) all $l \in A''$ are unit/pure in $\phi[A']$ after applying QBCP until completion.*

Clause and cube learning is carried out in function `analyze`. If $\phi[A] = \mathsf{F}$ then by rules *init*, *red*, and *res* the clause C where $UR(C[A]) = \emptyset$ is resolved with antecedent clauses to derive a learned clause C'.

If $\phi[A] = \mathsf{T}$, then a new learned cube is derived similarly to clause learning. However, rule *init* is special for cube learning in that cubes to be resolved later must be first derived from satisfying assignments. In contrast to that, clauses present in the input PCNF ϕ can be simply selected by rule *init.* QBCP is also applied to learned clauses and cubes. Related to unit clauses, a cube $ER(C[A]) = (l)$ containing the single literal l with $\mathsf{Q}(\Pi, l) = \forall$ is *unit* under A and existential reduction. Existential reduction is also applied in QBCP. Given a unit cube

$ER(C[A]) = (l)$, A is extended to $A := A \cup \{\bar{l}\}$ and C is recorded as the *antecedent cube* of the assignment $\{\bar{l}\}$. In cube learning, resolution by rule *res* is applied to cubes derived by rule *init* and to antecedent cubes.

Learned clauses (cubes) C' are constructed so that $UR(C'[A])$ ($ER(C'[A])$) is unit after backtracking. QCDCL terminates if the empty learned clause (cube) $C' = \emptyset$ is derived (`B == INVALID`). The derivations of learned clauses (cubes) up to $C' = \emptyset$ are a clause (cube) resolution proof of ϕ. The application of the rules of QRES is driven by the assignments generated in QCDCL.

In practice, the set θ of learned clauses is added conjunctively to $\phi = \Pi.\psi$ and we have $\phi \equiv_{sat} \Pi.(\psi \wedge \bigwedge_{C \in \theta} C)$. The set γ of learned cubes is added disjunctively to $\phi = \Pi.\psi$ and we have $\phi \equiv_{sat} \Pi.(\psi \vee (\bigvee_{C \in \gamma} C))$. These equivalence are due to the soundness of QRES. In general, a formula $\phi' = \Pi'.\psi'$ with $\psi' = \bigvee_{C \in \gamma} C$ in *prenex disjunctive normal form* can be derived by rule *init* so that $\phi \equiv_{sat} \phi'$ [8].

Preprocessing [3,7,11,21,26] aims at transforming the input PCNF ϕ into a simplified PCNF ϕ' with $\phi \equiv_{sat} \phi'$ so that ϕ' is solved faster than ϕ. In QCDCL, preprocessing can be applied once to ϕ before entering the `while`-loop (Fig. 1).

Inprocessing [13] combines preprocessing techniques and formula simplification under assignments which were fixed during a run of QCDCL. An assignment to a variable x is fixed if a unit clause (cube) $C = (l)$ with $\texttt{var}(l) = x$ is learned. Inprocessing can be applied after QBCP each time a unit clause (cube) is learned.

4 Improved Cube Learning by Dynamic QBCE

We take a closer look at cube learning by QRES. It is well known that in the worst case an exponential number of cubes must be derived by rule *init* even on PCNFs with a simple syntactic structure. The PCNFs in the following example are hard for QCDCL based on QRES. We develop a generalization of QRES which allows to solve these PCNFs easily by tightly integrating QBCE in QCDCL.

Example 1. Let $\Phi(n) = \exists z_1, z_1' \forall u_1 \exists y_1, \ldots, \exists z_n, z_n' \forall u_n \exists y_n. \bigwedge_{i=1}^{n} \left[\mathcal{C}_0(i) \wedge \mathcal{C}_1(i) \wedge \mathcal{C}_2(i)\right]$, where $\mathcal{C}_0(i) = (u_i \vee \bar{y}_i) \wedge (\bar{u}_i \vee y_i)$, $\mathcal{C}_1(i) = (z_i \vee u_i \vee \bar{y}_i) \wedge (z_i' \vee \bar{u}_i \vee y_i)$, and $\mathcal{C}_2(i) = (\bar{z}_i \vee \bar{u}_i \vee \bar{y}_i) \wedge (\bar{z}_i' \vee u_i \vee y_i)$ be a family of satisfiable PCNFs.

Clauses in $\mathcal{C}_0(i)$ encode the equivalence of u_i and y_i. In general, PCNFs of the form $\Psi(n) = \forall u_1 \exists y_1, \ldots, \forall u_n \exists y_n. \bigwedge_{i=1}^{n} \mathcal{C}_0(i)$ are typical examples where every cube resolution proof requires an exponential number of applications of rule *init* [12,16,18]. Since $\Psi(n)$ is a subformula of $\Phi(n)$, this also holds for every cube resolution proof of $\Phi(n)$. No clause is blocked in $\Phi(n)$, hence QBCE alone cannot solve $\Phi(n)$, in contrast to $\Psi(n)$ [12].

Consider $n = 1$ and $\Phi(n) = \exists z_1, z_1' \forall u_1 \exists y_1.(u_1 \vee \bar{y}_1) \wedge (\bar{u}_1 \vee y_1) \wedge (z_1 \vee u_1 \vee \bar{y}_1) \wedge (z_1' \vee \bar{u}_1 \vee y_1) \wedge (\bar{z}_1 \vee \bar{u}_1 \vee \bar{y}_1) \wedge (\bar{z}_1' \vee u_1 \vee y_1)$. By rule *init*, we derive $C_0 = (\bar{z}_1 \wedge \bar{z}_1' \wedge \bar{u}_1 \wedge \bar{y}_1)$ and $C_1 = (\bar{z}_1 \wedge \bar{z}_1' \wedge u_1 \wedge y_1)$. By existential reduction we get $C_2 = ER(C_0) = (\bar{z}_1 \wedge \bar{z}_1' \wedge \bar{u}_1)$ and $C_3 = ER(C_1) = (\bar{z}_1 \wedge \bar{z}_1' \wedge u_1)$. By resolving C_2 and C_3 we get $C_4 = (\bar{z}_1 \wedge \bar{z}_1')$ and finally $C_5 = ER(C_4) = \emptyset$.

The PCNFs $\Phi(n)$ in Example 1 can be solved by preprocessing by eliminating subsumed clauses and QBCE. In practice, however, preprocessing might not be fully applicable to QBF-based workflows involving advanced techniques such as, e.g., incremental QBF solving. Hence we aim at improving QCDCL with clause and cube learning also in the absence of preprocessing.

The application of rule *init* to derive cubes from satisfying assignments is also called *model generation* [8]. Model generation derives cubes C from a PCNF $\phi = \Pi.\psi$ such that $\phi[C] = \emptyset$. The PCNFs $\Phi(n)$ in Example 1 only have cube resolution proofs in QRES whose size is exponential in n since an exponential number of cubes must be derived by rule *init* [12,16,18]. Hence the run time of QCDCL with cube learning by QRES scales exponentially in n.

In the following, we generalize model generation by rule *init* by relaxing the condition $\phi[C] = \emptyset$. This way, we obtain a variant of QRES for cube learning which is exponentially stronger than the traditional variant from Definition 1. The stronger calculus allows for cube resolution proofs of the PCNFs $\Phi(n)$ in Example 1 whose size is polynomial in n.

To show the soundness of generalized model generation, we first show that a cube C which contains only variables assigned in decision making in QCDCL is derivable by QRES if $\phi[C]$ is satisfiable. To this end, we introduce the notion of cubes obtained from assignments generated in QCDCL.

Definition 4 (QCDCL Cube). *Given a QBF $\phi = \Pi.\psi$. The* QCDCL cube *C of QCDCL assignment A is defined by $C = (\bigwedge_{l \in A} l)$. Then $C = C' \cup C''$ where C' is the maximal subset of C such that $X_1 \cup \ldots \cup X_{i-1} \subset \mathsf{var}(C')$ and $C' \cap X_i \neq \emptyset$. The literals in C' are the first $|C'|$ consecutive variables of Π which are assigned, i.e., C' contains all the variables in C assigned in decision making.*[1] *The literals in C'' are assigned due to pure and unit literal detection and may occur anywhere in Π starting from X_{i+1}. For QCDCL cube C we further define $\mathsf{dec}(C) = C'$ and $\mathsf{der}(C) = C''$.*

Lemma 1. *Given the satisfiable PCNF $\phi = \Pi.\psi$ with $|\mathsf{var}(\Pi)| = n$ and a QCDCL cube C with $\mathsf{dec}(C) = C$. If $\phi[C]$ is satisfiable, then $\phi \vdash C$.*

Proof. We argue that $\phi \vdash C$ by induction over $k = n - m$ where $m = |C|$ and the rules of QRES shown in Definition 1.

If $k = 0$, then $\phi \vdash C$ by rule *init*. Consider $k > 0$. Let $l \in C$ with $\mathsf{var}(l) \in X_i$ be maximal in C w.r.t. $<_\Pi$. Let $h, \bar{h} \notin C$ and $\mathsf{var}(h) \in X_i$ if $X_i \backslash \mathsf{var}(C) \neq \emptyset$ and $\mathsf{var}(h) \in X_{i+1}$ otherwise.

Suppose that $\phi[C]$ is satisfiable. If $\mathsf{Q}(\Pi, h) = \forall$ then both $\phi[C \cup \{h\}]$ and $\phi[C \cup \{\bar{h}\}]$ are satisfiable. By induction hypothesis it holds that $\phi \vdash C \cup \{h\}$ and $\phi \vdash C \cup \{\bar{h}\}$, so $\phi \vdash C$ by the application of rule *res* (cf. Theorem 1 in [20]).

If $\mathsf{Q}(\Pi, h) = \exists$ then at least one of $\phi[C \cup \{h\}]$ and $\phi[C \cup \{\bar{h}\}]$ is satisfiable. W.l.o.g. assume that $\phi[C \cup \{h\}]$ is satisfiable. Then by induction hypothesis it holds that $\phi \vdash C \cup \{h\}$, so $\phi \vdash C$ by the application of rule *red*. □

[1] C' can also contain literals assigned by pure/unit literal detection, but as they are left to the maximal decision variable in the prefix, we treat them like decision variables.

Definition 5 (Generalized Model Generation). *Given a PCNF ϕ and a QCDCL assignment A according to Definition 3. If $\phi[A]$ is satisfiable, then the QCDCL cube $C = (\bigwedge_{l \in A} l)$ is obtained by* generalized model generation.

Condition $\phi[C] = \emptyset$ in case (2) of rule *init* is a special case of the condition in Definition 5 that $\phi[A]$ is satisfiable to obtain C. In general $\phi \nvdash C$ in QRES.

Example 2. Given a satisfiable PCNF $\phi = \exists x_1, x_2 \forall u_1 \exists x_3.(x_1 \vee u_1 \vee \bar{x}_3) \wedge (x_1 \vee u_1 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{u}_1 \wedge x_3) \wedge (x_2 \vee \bar{x}_3)$, where no variable is pure initially. Let $A = \{x_1\}$ by decision making. Then u_1 becomes pure and $\phi[A] = \exists x_2, x_3.(\bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3)$ is satisfiable under $A = \{x_1, u_1\}$, so $C = (\bigwedge_{l \in A} l) = (x_1 \wedge u_1)$ is obtained by generalized model generation. However, $\phi \nvdash C$ since rule *res* is not applicable as this would eliminate u_1. Further, any cube $C' \supset C$ derived by rule *init* contains also a literal of x_2 which cannot be reduced by rule *red*.

Theorem 1. *Given PCNF $\phi = \Pi.\psi$ and a QCDCL cube C obtained from ϕ by generalized model generation. Then it holds that $\Pi.\psi \equiv_{sat} \Pi.(\psi \vee C)$.*

Proof (Sketch). Let $C = C' \cup C''$ as defined in Definition 4 with $C' = \mathsf{dec}(C)$ and $C'' = \mathsf{der}(C)$. Recall that $\phi[C]$ is satisfiable, because C is obtained by generalized model generation.

First, assume that $\phi[C']$ is unsatisfiable. The literals in C'' are assigned according to pure and unit literal detection in $\phi[C']$ which is sound. Therefore, these assignments do not change the satisfiability status of the formula. Hence, $\phi[C']$ has to be satisfiable.

According to Lemma 1, it holds that $\Pi.\psi \vdash C'$ and due to the soundness of QRES it holds that $\Pi.\psi \equiv_{sat} \Pi.(\psi \vee C')$. Because of subsumption, it holds that $\Pi.\psi \equiv_{sat} \Pi.(\psi \vee (C' \wedge C''))$. □

Corollary 1. *By Theorem 1, a cube C obtained from PCNF $\Pi.\psi$ by generalized model generation can be used as a learned cube in QCDCL.*

Definition 6. *Let $\phi = \Pi.\psi$ be a PCNF. The* Q-resolution calculus with generalized model generation (QRES-GMG) *is obtained by replacing condition $\phi[C] = \emptyset$ in case (2) of rule init in Definition 1 by the condition that $\phi[C]$ is satisfiable.*

QRES-GMG can be used as a proof system underlying QCDCL to derive learned clauses and cubes. For generalized model generation as part of rule *init*, it is necessary to check whether $\phi[A]$ is satisfiable based on the current assignment A enumerated in QCDCL. Since $\phi[A]$ is a PCNF in general, such check is as hard as solving the original PCNF ϕ (i.e., PSPACE-complete).

In order to combine QRES-GMG and QCDCL in practice, we apply QBCE *dynamically* to $\phi[A]$, i.e., with respect to the current assignment A. If all clauses in $\phi[A]$ are blocked then QBCE reduces $\phi[A]$ to the empty matrix, thus showing that $\phi[A]$ is satisfiable in time which is *polynomial* in the size of $\phi[A]$. This way, we apply QBCE as an incomplete decision procedure inside QCDCL to efficiently check if $\phi[A]$ is satisfiable. To this end, in general any sound decision procedure can be applied. However, QBCE is appealing since it only removes clauses and can be implemented using data structures which fit in the QCDCL framework.

Generalized model generation in QRES-GMG is related to *sign abstraction* (Proposition 7 in [16]). Based on a previously derived learned cube C', sign abstraction allows to detect whether $\phi[A]$ is satisfiable in polynomial time based on the sets of clauses satisfied by C' and by A. However, our approach to checking $\phi[A]$ based on dynamic QBCE is independent from previously learned cubes.

In contrast to QRES, the PCNFs $\Phi(n)$ in Example 1 have short cube resolution proofs in QRES-GMG. Hence QCDCL with QRES-GMG and dynamic QBCE allows for more powerful cube learning than with QRES.

Example 3 (Continues Example 1). For $n = 2$ consider $\Phi(n) = \exists z_1, z_1' \forall u_1 \exists y_1, \exists z_2, \exists z_2' \forall u_2 \exists y_2 . \mathcal{C}_0(1) \wedge \mathcal{C}_1(1) \wedge \mathcal{C}_2(1) \wedge \mathcal{C}_0(2) \wedge \mathcal{C}_1(2) \wedge \mathcal{C}_2(2)$. We solve $\Phi(n)$ by QCDCL with QRES-GMG and dynamic QBCE. We start with the empty assignment $A = \emptyset$. No clause is blocked in the PCNF $\Phi(n)[A]$ and hence rule *init* is not applicable. By decision making we assign variables from left to right in prefix ordering and extend A to $A = A \cup \{\bar{z}_1, \bar{z}_1'\}$. The clauses in the subformula $(\mathcal{C}_0(1) \wedge \mathcal{C}_1(1))[A] = (u_1 \vee \bar{y}_1) \wedge (\bar{u}_1 \vee y_1)$ of $\Phi(n)[A]$ are blocked since all clauses in $\mathcal{C}_2(1)[A]$ are satisfied under A.

Before making further assignments, the PCNF $\Phi(n)$ is simplified to $\Phi(n)' = \exists z_2, z_2' \forall u_2 \exists y_2 . \mathcal{C}_0(2) \wedge \mathcal{C}_1(2) \wedge \mathcal{C}_2(2)$ under A and QBCE. No clause is blocked in $\Phi(n)'[A]$ and A is extended to $A \cup \{\bar{z}_1, \bar{z}_1', \bar{z}_2, \bar{z}_2'\}$. Like before, clauses in the subformula $(\mathcal{C}_0(2) \wedge \mathcal{C}_1(2))[A] = (u_2 \vee \bar{y}_2) \wedge (\bar{u}_2 \vee y_2)$ of $\Phi(n)'[A]$ are blocked since all clauses in $\mathcal{C}_2(2)[A]$ are satisfied under A. We have $\Phi(n)[A] = \emptyset$ under $A = \{\bar{z}_1, \bar{z}_1', \bar{z}_2, \bar{z}_2'\}$ and QBCE. By rule *init* of QRES-GMG, we derive the learned cube $C = (\bar{z}_1 \wedge \bar{z}_1' \wedge \bar{z}_2 \wedge \bar{z}_2')$ and finally the empty cube $C' = ER(C) = \emptyset$ by existential reduction, after which QCDCL terminates.

Note that on the PCNFs $\Phi(n)$ from Example 1, for *any* value of n QCDCL with QRES-GMG based on dynamic QBCE learns *exactly* one cube by rule *init* which is reduced to the empty cube immediately. Hence in this special case the actual run time of QCDCL depends on how efficiently QBCE is applied to $\Phi(n)$.

5 Integrating Dynamic QBCE in QCDCL

We implemented QCDCL with clause and cube learning based on QRES-GMG in our solver DepQBF.[2] DepQBF is a QCDCL-based solver which originally relies on QRES [8,16,28]. For efficient generalized model generation in QRES-GMG, we apply QBCE dynamically as illustrated by Example 3.

QBCE is carried out eagerly as part of QBCP (function `qbcp` in Fig. 1). After the current assignment A has been extended in QBCP by unit and pure literal detection, QBCE is applied to $\phi[A]$ until completion. If all clauses in $\phi[A]$ are blocked, then a cube is learned by rule *init* in QRES-GMG. Otherwise, A is further extended by decision making and again QBCP including QBCE is applied. All clauses containing a literal of a variable x may be blocked in $\phi[A]$ and x may be removed from the prefix of $\phi[A]$. Hence in decision making variables are selected from the left end of the prefix of $\phi[A]$ simplified under A and QBCE.

[2] http://lonsing.github.io/depqbf/.

Clause learning works as in traditional QCDCL based on QRES. Clauses currently blocked with respect to A are ignored when it comes to the detection of unit clauses and empty clauses in QBCP. Consequently, such blocked clauses are not used to derive learned clauses. However, since QBCE preserves unsatisfiability, QCDCL will eventually find a clause resolution proof of ϕ if ϕ is unsatisfiable even if dynamic QBCE effectively removes blocked clauses. Hence we apply QBCE in a fully dynamic way where QCDCL with clause and cube learning operates on the PCNF $\phi[A]$ simplified under A and QBCE. QBCE is applied only to clauses in the input PCNF ϕ, not to learned ones.

5.1 Witness Clauses for Efficient Dynamic QBCE

Dynamic QBCE is part of QBCP and hence is applied frequently in QCDCL. Our implementation of dynamic QBCE relies on watched data structures like QBCP [6]. In the following, let A be the current assignment in QCDCL.

Dynamic QBCE is carried out based on a *working set* of pairs (C, l) of a clause C and a literal $l \in C$ to be checked whether l is a blocking literal in C. The working set is fully processed as part of QBCP. Variable assignments made in QBCP might trigger further applications of QBCE, which causes the working set to be filled based on watched data structures as described in the following.

For each clause C in the input PCNF ϕ a *notification list* is maintained. The notification list of C contains pairs (C', l') of clauses C' and an existential literal $l' \in C'$ such that $\bar{l'} \in C$ and C is a witness that $l' \in C'$ is not a blocking literal by Definition 2 in C' under A. The witness C is neither blocked nor satisfied under A. If C becomes either blocked or satisfied, then the pair (C', l') is put in the working set to be checked whether $l' \in C'$ is a blocking literal in C'.

Each variable v has two lists L_v and $L_{\bar{v}}$ containing clauses $C \in \phi$ with a positive and negative literal of v, respectively. Each clause C in L_v or $L_{\bar{v}}$ is a witness that some other clause $C' \in \phi$ is not blocked with a blocking literal $l \in C'$, $\bar{l} \in C$, and $\mathsf{var}(l) = v$. If v is assigned true (false), then previously non-satisfied witness clauses $C \in L_v$ ($C \in L_{\bar{v}}$) are satisfied. All pairs (C', l') in the notification list of the now satisfied witness C are put in the working set.

Watched data structures based on witness clauses allow to carry out dynamic QBCE precisely when a witness clause becomes blocked or satisfied under A. Superfluous checks of Definition 2 are entirely avoided.

5.2 Dynamic QBCE Limits

The input PCNF ϕ may contain clauses with a large number of literals or variables whose literals appear in a large number of clauses (called *occurrences*). In these cases, the performance of dynamic QBCE may deteriorate with respect to run time and memory footprint since the working set and the notification lists become prohibitively large. To control the computational costs of dynamic QBCE, we implemented a limit *max_lits* on the size $|C|$ (i.e. number of literals) of a clause and a limit *max_occs* on the number of occurrences of variables.

For all pairs (C, l) ever put in the working set it holds that $|C| \leq \mathit{max\_lits}$. Clauses C whose size $|C|$ exceeds $\mathit{max\_lits}$ are permanently ignored in dynamic QBCE. Additionally, the size $|C'|$ of each occurrence C' with $\bar{l} \in C'$ of variable $\mathsf{var}(l)$ must not exceed $\mathit{max\_lits}$. This way, literals of variables with occurrences larger than $\mathit{max\_lits}$ are never checked as potential blocking literals. Limit $\mathit{max\_lits}$ allows to avoid inspecting large clauses when checking Definition 2.

For all pairs (C, l) ever put in the QBCE working set the number of occurrences of literal $\bar{l}$ does not exceed $\mathit{max\_occs}$. This way, no more than $\mathit{max\_occs}$ occurrences of $\mathsf{var}(\bar{l})$ have to be inspected when checking whether $l \in C$ is a blocking literal in C by Definition 2.

QCDCL based on QRES-GMG is sound in the presence of limits $\mathit{max\_lits}$ and $\mathit{max\_occs}$. For generalized model generation, clauses ignored in dynamic QBCE due to the limits must be satisfied under the current assignment A.

In our implementation of dynamic QBCE we used limits of $\mathit{max\_lits} = 50$ and $\mathit{max\_occs} = 50$ which we determined empirically.

6 Experiments

We evaluated our implementation of QCDCL with QRES-GMG based on dynamic QBCE in the solver DepQBF. To this end, we compared three variants of DepQBF. The plain version of DepQBF (no-qbce) is based on traditional QCDCL with QRES. As a first step towards dynamic QBCE, we implemented QBCE as inprocessing in DepQBF (qbce-inp) where the input PCNF ϕ is simplified by QBCE only with respect to fixed assignment due to learned *unit clauses* and *unit cubes*. This variant of DepQBF is still based on QRES. Finally, we implemented QCDCL with QRES-GMG based on fully dynamic QBCE (qbce-dyn) as presented in Sect. 5. To focus on dynamic QBCE, we used the linear quantifier ordering given by the prefix of the input PCNF ϕ and hence disabled advanced analysis of variable dependencies [19] in all variants of DepQBF.

Further, we consider the solvers GhostQ (ghostq) and RAReQS (rareqs) which were both among the winning solvers of the QBF Gallery 2014 and which are publicly available. All experiments reported in the following were run on an AMD Opteron 6238 at 2.6 GHz under 64-bit Linux with time and memory limits of 1800 s and 7 GB. We use the benchmarks in the applications set of the QBF Gallery consisting of 735 formulas.[3] We do not consider structural solvers because the benchmarks are not available in a structural, non-CNF format.

First, we consider the original applications set without preprocessing. The results are shown in Table 1 and Fig. 2. The combinations of QBCE and QCDCL in qbce-inp and qbce-dyn considerably outperform DepQBF (no-qbce) by solved instances and run time. Moreover, qbce-dyn performs best among the variants of DepQBF and outperforms RAReQS, an expansion-based solver. Whereas the variants of DepQBF did not run out of memory, RAReQS did on 115 instances.

In order to evaluate the impact of preprocessing on solver performance, we applied the preprocessor Bloqqer [11] prior to solving. The results are shown in

[3] http://qbf.satisfiability.org/gallery.

Table 1. Total solved instances, solved unsatisfiable, and solved satisfiable ones of the original applications set of the QBF Gallery 2014 without preprocessing. Reported times are total run times including time outs. Running out of memory is counted as a time out.

Solver	Solved	Unsat	Sat	Time
qbce-dyn	441	222	219	573,142
rareqs	414	272	142	611,742
qbce-inp	360	161	199	735,073
ghostq	347	166	181	752,950
no-qbce	278	128	150	880,485

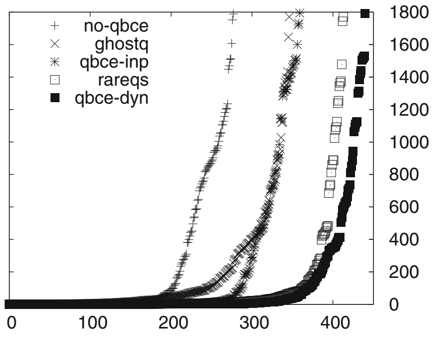

Fig. 2. Sorted run times (y-axis) of instances (x-axis) related to Table 1.

Table 2 and Fig. 4. RAReQS and DepQBF (no-qbce) benefit from preprocessing. RAReQS ran out of memory on 34 instances. DepQBF (no-qbce) solves as many formulas as qbce-inp. Among the variants of DepQBF, qbce-dyn still solves the largest number of instances. However, preprocessing has an overall negative effect on qbce-dyn as only 405 instances are solved with preprocessing compared to 441 instances without preprocessing (Tables 1 and 2 are comparable since no instance was solved by preprocessing).

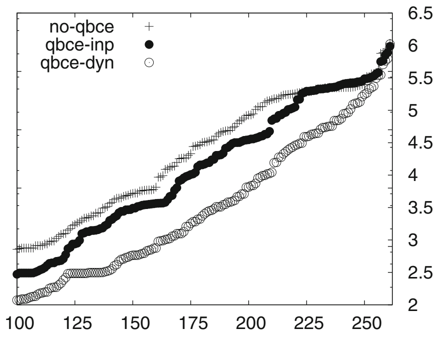

Fig. 3. Related to Table 1: sorted numbers of backtracks (log_{10} scale y-axis) by DepQBF, qbce-inp, and qbce-dyn on 262 instances solved by all three (x-axis).

Preprocessing by Bloqqer, which includes QBCE among other techniques, changes the formula structure such that dynamic QBCE (qbce-dyn) does not pay off any more. Although RAReQS solves 142 instances more, DepQBF with dynamic QBCE solves 25 instances not solved by RAReQS.

In additional experiments, we found out that the actual selection of techniques applied for preprocessing by Bloqqer has a considerable impact on the number of instances solved by RAReQS and DepQBF with dynamic QBCE (qbce-dyn). Limited preprocessing is beneficial for qbce-dyn whereas it has a negative impact on RAReQS compared to full preprocessing (Table 2). We preprocessed the applications set by Bloqqer using only QBCE and expansion of universal variables [3]. On this preprocessed set, RAReQS solves only 471 instances compared to 547 with full preprocessing (Table 2). In contrast to that, qbce-dyn solves 463 instances compared to 405 with full preprocessing. Hence limited preprocessing reduces the gap between

Table 2. Like Table 1, but on the applications set of the QBF Gallery 2014 with preprocessing by Bloqqer prior to solving. No instance was solved in preprocessing.

Solver	Solved	Unsat	Sat	Time
rareqs	547	314	233	379,916
qbce-dyn	405	201	204	624,719
no-qbce	390	205	185	651,909
qbce-inp	390	205	185	655,329
ghostq	350	176	174	739,294

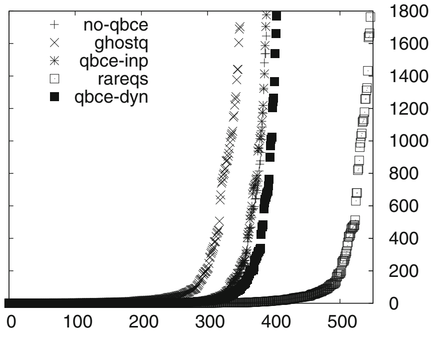

Fig. 4. Sorted run times (y-axis) of instances (x-axis) related to Table 2.

RAReQS and qbce-dyn from 142 (Table 2) to eight solved instances. This is due to the fact that RAReQS performs worse with limited preprocessing than with full preprocessing.

In addition to solved instances, the benefits of dynamic QBCE are also reflected by backtracks in QCDCL. DepQBF with dynamic QBCE backtracks less frequently than the other variants of DepQBF. Figure 3 illustrates the numbers of backtracks on those 262 instances which were solved by all three variants of DepQBF. The average (median) number of backtracks is 160,597 (3,119) by no-qbce, 133,919 (1,793) by qbce-inp, and 66,372 (350) by qbce-dyn. We made similar observations when comparing only qbce-inp and qbce-dyn.

A comparison of no-qbce and qbce-dyn in Table 3 shows that dynamic QBCE results in fewer *redundant* backtracks in QCDCL with respect to resolution proofs. For example, when solving an unsatisfiable PCNF ϕ by QCDCL, we consider backtracks from satisfiable subcases (i.e., `R == SAT` in Fig. 1) redundant because these backtracks result in learned cubes. However, learned cubes are irrelevant to the clause resolution proof of ϕ produced by QCDCL. On unsatisfiable instances, the numbers of redundant backtracks by no-qbce and qbce-dyn differ by a factor of 24 (54,078 vs. 2,199). These results indicate the potential

Table 3. Related to Table 1: average run time in seconds (T) and number of backtracks resulting from satisfiable (SB) and unsatisfiable (UB) subcases in QCDCL (i.e. `R == UNSAT/SAT` in Fig. 1) on instances solved by both DepQBF without QBCE (no-qbce) and DepQBF with dynamic QBCE (qbce-dyn). Statistics are shown based on all solved instances (265) and separately for solved satisfiable (141) and unsatisfiable (124) ones.

	ALL (265)			*SAT (141)*			*UNSAT (124)*		
	T	SB	UB	T	SB	UB	T	SB	UB
no-qbce	181	59,044	103,080	81	63,412	17,356	295	54,078	200,557
qbce-dyn	80	22,805	42,969	51	40,927	15,979	114	2,199	73,660

benefits of generalized model generation in QRES-GMG for deriving learned cubes to prune the search space tackled by QCDCL. On satisfiable instances, the difference in redundant backtracks is less pronounced (17,356 vs. 15,979).

We also ran experiments on the preprocessing and QBFLIB tracks of the QBF Gallery 2014. Dynamic QBCE (qbce-dyn) does not pay off on the massively preprocessed instances in the preprocessing track. There, RAReQS solves the largest number of instances (107) and qbce-dyn solves 95 compared to 101 solved by no-qbce and qbce-inp. On the QBFLIB track, both qbce-inp (108 solved) and qbce-dyn (104) clearly outperform no-qbce (83) and RAReQS (80), where GhostQ solves 139 instances. On the QBFLIB track with full preprocessing by Bloqqer, the performance of qbce-dyn (131 solved), no-qbce (130), and qbce-inp (129) is close to each other, where RAReQS solves 134 instances.

7 Conclusion

We presented dynamic blocked clause elimination (QBCE) in QCDCL-based QBF solvers as an approach to overcome the bias towards unsatisfiability in solving that is due to the CNF structure of QBFs. Thereby, QBCE is applied eagerly to the QBF interpreted under the assignments generated in QCDCL. Dynamic QBCE results in a variant of cube learning by QRES-GMG in QCDCL which is exponentially stronger than the traditional variant.

On application instances, we observed a considerable performance boost with dynamic QBCE—despite its computational overhead—in terms of solved instances, run time, and backtracks. Without preprocessing, our approach outperforms expansion-based QBF solving. Depending on the selection of techniques, preprocessing may have a negative impact on the performance of dynamic QBCE since formula structure is blurred. However, dynamic QBCE may improve the performance of QCDCL solvers in workflows involving incremental solving, which cannot yet be combined with full-scale preprocessing. Dynamic QBCE is compatible with incremental solving, in contrast to expansion-based solving.

Our approach is extensible in that techniques other than QBCE like bounded variable elimination or expansion can be applied dynamically for generalized model generation. Further, dynamic QBCE can be readily combined with any variant of Q-resolution like QU-resolution [24] and long-distance resolution [28] as part of rule *res* in QRES-GMG. We also aim at combining our approach with full generation of proofs and certificates.

References

1. Ansótegui, C., Gomes, C.P., Selman, B.: The achilles' heel of QBF. In: AAAI/IAAI, pp. 275–281. AAAI Press/The MIT Press (2005)
2. Benedetti, M., Mangassarian, H.: QBF-based formal verification: experience and perspectives. JSAT **5**(1–4), 133–191 (2008)
3. Bubeck, U., Kleine Büning, H.: Bounded universal expansion for preprocessing QBF. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 244–257. Springer, Heidelberg (2007)

4. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
5. Egly, U., Seidl, M., Woltran, S.: A solver for QBFs in negation normal form. Constraints **14**(1), 38–79 (2009)
6. Gent, I.P., Giunchiglia, E., Narizzano, M., Rowley, A.G.D., Tacchella, A.: Watched data structures for QBF solvers. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 25–36. Springer, Heidelberg (2004)
7. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: an effective preprocessor for QBFs based on equivalence reasoning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 85–98. Springer, Heidelberg (2010)
8. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified boolean formulas. JAIR **26**, 371–416 (2006)
9. Goultiaeva, A., Bacchus, F.: Exploiting circuit representations in QBF solving. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 333–339. Springer, Heidelberg (2010)
10. Goultiaeva, A., Seidl, M., Biere, A.: Bridging the gap between dual propagation and CNF-based QBF solving. In: Järvisalo, M., Van Gelder, A. (ed.) DATE, pp. 811–814. ACM (2013)
11. Heule, M., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. JAIR **53**, 127–168 (2015)
12. Janota, M., Grigore, R., Marques-Silva, J.: On QBF proofs and preprocessing. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 473–489. Springer, Heidelberg (2013)
13. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
14. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified boolean formulas. Inf. Comput. **117**(1), 12–18 (1995)
15. Klieber, W., Sapra, S., Gao, S., Clarke, E.: A non-prenex, non-clausal QBF solver with game-state learning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 128–142. Springer, Heidelberg (2010)
16. Letz, R.: Lemma and model caching in decision procedures for quantified boolean formulas. In: Egly, U., Fermüller, C. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, pp. 160–175. Springer, Heidelberg (2002)
17. Pulina, L., Tacchella, A.: A structural approach to reasoning with quantified boolean formulas. In: IJCAI, pp. 596–602 (2009)
18. Ramesh, A., Becker, G., Murray, N.V.: CNF and DNF considered harmful for computing prime implicants/implicates. JAIR **18**(3), 337–356 (1997)
19. Samer, M., Szeider, S.: Backdoor sets of quantified boolean formulas. JAR **42**(1), 77–97 (2009)
20. Samulowitz, H., Bacchus, F.: Dynamically partitioning for solving QBF. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 215–229. Springer, Heidelberg (2007)
21. Samulowitz, H., Davies, J., Bacchus, F.: Preprocessing QBF. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 514–529. Springer, Heidelberg (2006)
22. Schaefer, T.J.: On the complexity of some two-person perfect-information games. J. Comput. Syst. Sci. **16**(2), 185–225 (1978)
23. Silva, J.P.M., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999)

24. Van Gelder, A.: Contributions to the theory of practical quantified boolean formula solving. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 647–663. Springer, Heidelberg (2012)
25. Van Gelder, A.: Primal and dual encoding from applications into quantified boolean formulas. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 694–707. Springer, Heidelberg (2013)
26. Van Gelder, A., Wood, S.B., Lonsing, F.: Extended failed-literal preprocessing for quantified boolean formulas. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 86–99. Springer, Heidelberg (2012)
27. Zhang, L.: Solving QBF by combining conjunctive and disjunctive normal forms. In: Dustdar, S., Schall, D., Skopik, F., Juszczyk, L., Psaier, H. (eds.) AAAI/IAAI, pp. 143–150. AAAI Press (2006)
28. Zhang, L., Malik, S.: Conflict driven learning in a quantified boolean satisfiability solver. In: ICCAD, pp. 442–449. ACM/IEEE Computer Society (2002)