Efficient Clause Learning for Quantified Boolean Formulas via QBF Pseudo Unit Propagation*

Florian Lonsing[1], Uwe Egly[1], and Allen Van Gelder[2]

[1] Technische Universität Wien
`http://www.kr.tuwien.ac.at/staff/{lonsing,egly}`
[2] University of California, Santa Cruz, `http://www.cse.ucsc.edu/~avg`

Abstract. Recent solvers for quantified boolean formulas (QBF) use a clause learning method based on a procedure proposed by Giunchiglia et al. (JAIR 2006), which avoids creating tautological clauses. Recently, an exponential worst case for this procedure has been shown by Van Gelder (CP 2012). That paper introduced QBF Pseudo Unit Propagation (QPUP) for non-tautological clause learning in a limited setting and showed that its worst case is theoretically polynomial, although it might be impractical in a high-performance QBF solver, due to excessive time and space consumption. No implementation was reported.
We describe an enhanced version of QPUP learning that is practical to incorporate into high-performance QBF solvers, being compatible with pure-literal rules and dependency schemes. It can be used for proving in a concise format that a QBF formula is either unsatisfiable or satisfiable (working on both proofs in tandem). A lazy version of QPUP permits non-tautological clauses to be learned without actually carrying out resolutions, but this version is unable to produce proofs.
Experimental results show that QPUP is somewhat faster than the previous non-tautological clause learning procedure on benchmarks from QBFEVAL-12-SR.

1 Introduction

Solvers for Quantified Boolean Formulas (QBFs) are rapidly increasing in strength, partly due to increased understanding of how to incorporate conflict-driven clause learning (CDCL), which found great practical success in propositional satisfiability. Several current solvers are patterned after the Q-resolution method described by Giunchiglia *et al.* [3]. A thorough survey of the field through 2005 may be found in this paper.

We present a formal framework for solvers to search for proofs that an instance is true or false in tandem, and to use information found in one proof search to assist in the other proof search. The framework is closely related to the work of Zhang and Malik [19], and Giunchiglia *et al.* [3], but uses the idea of a *Complete Guard Formula* and a closely related *Basic Guard Formula* to prove some key properties straightforwardly. This approach differs from that used for `CirQit2` [4] and `GhostQ` [7] in that those solvers use a representation in which the entire negated formula is known, whereas the guard formula is only partially known.

This paper then presents a prototype implementation of QBF Pseudo Unit Propagation (QPUP) non-tautological clause learning. The main innovation is to identify a cut

* The first two authors are supported by the Austrian Science Fund under grant S11409-N23.

in the conflict graph such that the learned clause associated with the cut will be *non-tautological* and *asserting*: that is, after backtracking, the learned clause will enable a new literal to be implied immediately. After the cut is identified, resolution proceeds from clauses on the boundary of the cut toward the conflicting clause, in an order such that tautologies cannot occur. Previous techniques performed resolutions beginning at the conflicting clause and working back toward the boundary of the cut, without ever explicitly defining the cut. When the latter order is used, tautologies can occur and require possibly expensive special treatment (exponential time in the worst case).

All clauses derived during the QPUP procedure satisfy an invariant that permits an asserting learned clause to be constructed lazily, without actually performing the resolutions. This shortcut saves substantial clause-learning time, but does not permit a detailed Q-resolution proof to be output as a by-product, and occasionally constructs a slightly weaker learned clause.

An implementation of QPUP learning was incorporated in the open-source solver `DepQBF` [8, 10, 9]. Experimental results (Section 6) illustrate the potential of QPUP learning compared to traditional clause learning in terms of more solved formulas, fewer backtracks and reduced run times.[3]

2 Preliminaries

In this section, we collect specific notation for later use. In general, *quantified boolean formulas* (QBFs) generalize propositional formulas by adding universal and existential quantification of boolean variables. See [6] for a thorough introduction. For this paper QBFs are in prenex conjunctive normal form (PCNF): $\Psi = \overrightarrow{Q}\,.\,\mathcal{F}$ consists of prenex $\overrightarrow{Q}$ and clause-matrix $\mathcal{F}$ (the *original* clauses). The prenex $\overrightarrow{Q}$ is partitioned into maximal contiguous subsequences of the same quantifier type, called *quantifier blocks*. Each quantifier block has a different *qdepth* with the outermost block having *qdepth* $= 1$. If p and q are variables of opposite quantifier types, we say that q is ***inner*** to p if *qdepth*$(p) <$ *qdepth*(q).

Clauses may be written as literals enclosed in square brackets (e.g., $[p, q, \overline{r}]$), and $[]$ denotes the empty clause. Where the context permits, letters e and others near the beginning of the alphabet denote existential literals, while letters u and others near the end of the alphabet denote universal literals. Letters like p, q, r denote literals of unspecified quantifier type. We use $\perp$ to denote the constant false in the role of a literal. The variable underlying literal p is denoted by $|p|$.

A *closed* QBF (all variable occurrences are quantified) evaluates to either 0 (false) or 1 (true), as defined by induction on its principal operator: **(1)** $(\exists\, x\, \phi(x)) = 1$ iff $\phi(0) = 1$ or $\phi(1) = 1$. **(2)** $(\forall\, x\, \phi(x)) = 0$ iff $\phi(0) = 0$ or $\phi(1) = 0$. **(3)** Other operators have the same semantics as in propositional logic. This definition emphasizes the connection of QBF to two-person games with complete information, in which player E (Existential) tries to set existential variables to make the QBF evaluate to 1, and player A (Universal) tries to set universal variables to make the QBF evaluate to 0.

[3] Please visit http://www.kr.tuwien.ac.at/staff/lonsing/sat13submission.tar.gz for a longer version of this paper, binaries, and some logs.

Players set their variable when it is outermost, or for non-prenex, when it is the root of a subformula (see [7] for more details). Only one player has a winning strategy.

The proof system known as *Q-resolution* consists of two operations, *resolution* and *universal reduction* (some papers combine them into one operation). Q-resolution is of central importance for QBFs because it is a sound and complete proof system [5].

Definition 2.1 ***Resolution*** is defined as usual. The *resolvent* of C_1 and C_2 on existential e is denoted as $\mathsf{res}_q(C_1, C_2) = [(C_1 - \overline{e}), (C_2 - e)]$ or as $C_1 *_e C_2$. (We drop set-forming braces around singleton sets where obvious.) The resolvent must be non-tautologous for Q-resolution. ***Universal reduction*** is special to QBF. The notation is $\mathsf{unrd}_q(C_3) = (C_3 - u)$, where C_3 is non-tautologous and u is tailing for C_3. Here, a universal literal u is said to be ***tailing*** for C_3 if no existential literal in C_3 is inner to u. A postfix operator notation for the same expression is $C_3\, \varDelta_u$. Performing all possible universal reductions on C_3 is denoted by $\mathsf{unrd}_*(C_3)$ or $C_3\, \varDelta_*$, and the resulting clause is said to be ***fully reduced***. The operators are left-associative, like $+$ and $-$, so that compound expressions without parentheses can be read left to right.

A ***Q-derivation*** of a clause is a proof using the Q-resolution operations; a ***Q-refutation*** is a Q-derivation of the empty clause. $\square$

Definition 2.2 An ***assignment*** (sometimes called a *partial assignment*) is a partial function from variables to truth values, and is usually represented as the set of literals that it maps to true. A ***total assignment*** assigns a truth value to every variable. Assignments are denoted by σ, τ, etc. Applications of σ to logical expressions are denoted by $q\lceil_\sigma$, $C\lceil_\sigma$, $\mathcal{F}\lceil_\sigma$, etc., and consist of replacing assigned variables in the expression by their truth values in σ, then simplifying with truth-value identities (but not propagating unit clauses). If σ assigns variables that are quantified in $\varPsi$, those quantifiers are deleted in $\varPsi\lceil_\sigma$, and their variables receive the assignment specified by σ. $\square$

3 Guard Formulas

This section describes *guard formulas*, and states their main properties that are important for QBF solving. We begin with some terminology.

Definition 3.1 Let the initial PCNF formula be $\varPsi = \overrightarrow{Q}.\,\mathcal{F}$, where $\overrightarrow{Q}$ is the quantifier prefix and $\mathcal{F}$ is the matrix of clauses. A ***consistent minimal hitting set*** **(cmhs)** for a set of clauses $\mathcal{F}$ is a partial assignment σ (regarded as a consistent set of literals) such that every clause $C \in \mathcal{F}$ is satisfied by σ and no proper subset of σ has this property. Note that $\mathcal{F}$ has no hitting set if it is propositionally unsatisfiable.

Let $\widetilde{Q}$ be the same as $\overrightarrow{Q}$ except that the quantifier type of each variable is inverted. The ***Complete Guard Formula*** for $\varPsi$ is the PCNF $\varGamma_* = \widetilde{Q}.\,\mathcal{G}_*(\mathcal{F})$, where $\mathcal{G}_*(\mathcal{F})$ is the set of clauses defined as follows:

$$C \in \mathcal{G}_*(\mathcal{F}) \text{ if and only if } \neg(C) \text{ is a cmhs for } \mathcal{F} \tag{1}$$

Here, $\neg(C)$ is the partial assignment consisting of the negations of all literals in C. (The negation of a clause is often called a cube.) $\square$

Ψ	$\forall u$	$\exists d$	$\exists a$	$\forall v$	$\exists b$	$\exists c$	$\exists e$
C_1	u	$\overline{d}$	a				
C_2	$\overline{u}$		a	$\overline{v}$		$\overline{c}$	$\overline{e}$
C_3	$\overline{u}$						e
C_4		$\overline{d}$	a	v	$\overline{b}$		
C_5		d	$\overline{a}$	$\overline{v}$	b		
C_6		d				c	
C_7			$\overline{a}$	v	$\overline{b}$	$\overline{c}$	
C_8				v	b		
C_9				$\overline{v}$	$\overline{b}$		
C_{10}				v		c	

Γ_*	$\exists u$	$\forall d$	$\forall a$	$\exists v$	$\forall b$	$\forall c$	$\forall e$
D_1	u	d	a	v	$\overline{b}$	$\overline{c}$	
D_2	u	d	a	$\overline{v}$	b	$\overline{c}$	
D_3	$\overline{u}$	$\overline{d}$		$\overline{v}$	b	c	$\overline{e}$
D_4		$\overline{d}$	$\overline{a}$	$\overline{v}$	b		$\overline{e}$
D_5		d	a	v	$\overline{b}$	$\overline{c}$	$\overline{e}$

Γ_B	$\exists u$	$\forall d$	$\forall a$	$\exists v$	$\forall b$	$\forall c$	$\forall e$
D_1	u	d	a	v			
D_2	u	d	a	$\overline{v}$			
D_3	$\overline{u}$	$\overline{d}$		$\overline{v}$			
D_4		$\overline{d}$	$\overline{a}$	$\overline{v}$			
D_5		d	a	v			

Fig. 1. QCNFs Ψ, Γ_*, and Γ_B, discussed in Example 3.6 and later examples.

Since Γ_* is a PCNF we may exploit the soundness and completeness of Q-resolution for PCNF refutations. In general discussions we call Ψ the *original formula* and call Γ_* and related formulas *guard formulas*.

Definition 3.2 If two propositional formulas $\mathcal{F}_1$ and $\mathcal{F}_2$ evaluate to the same truth value for every total assignment, then $\mathcal{F}_1 \equiv \mathcal{F}_2$, read as $\mathcal{F}_1$ and $\mathcal{F}_2$ are ***propositionally equivalent***. If every total assignment that satisfies $\mathcal{F}_1$ also satisfies $\mathcal{F}_2$, then $\mathcal{F}_1 \models \mathcal{F}_2$, read as $\mathcal{F}_1$ ***logically implies*** $\mathcal{F}_2$. $\square$

The idea of cubes is familiar in QBF, but Γ_* has this important property, which has not been enunciated before, as far as we know (proofs are omitted to save space):

Lemma 3.3 With the notation of Definition 3.1: **(A)** $\mathcal{G}_*(\mathcal{F}) \equiv \neg\mathcal{F}$; **(B)** Γ_* has the opposite truth value from Ψ. $\blacksquare$

Definition 3.4 Let Ψ and Γ_* and $\mathcal{F}$ and $\mathcal{G}_*$ be as defined in Definition 3.1. The ***Basic Guard Formula*** for Ψ is the PCNF $\Gamma_B = \widetilde{Q}.\, \mathcal{G}_B(\mathcal{F})$, where $\mathcal{G}_B(\mathcal{F})$ is the set of clauses arising by performing all possible universal reductions on clauses in $\mathcal{G}_*(\mathcal{F})$. (Recall that the quantifier type of each variable in $\widetilde{Q}$ is the opposite of its quantifier type in $\overrightarrow{Q}$.) $\square$

Lemma 3.5 With the notation of Definition 3.4: **(A)** Γ_B has the opposite truth value from Ψ; **(B)** Γ_B has a Q-refutation if and only if the truth value of Ψ is true; **(C)** For any $C \in \mathcal{G}_B(\mathcal{F})$, $\neg(C)$ can be extended to a cmhs for $\mathcal{F}$ by adding only literals e_i such that e_i is inner to some $u \in \neg(C)$, where e_i is existential and u is universal in $\overrightarrow{Q}$. $\blacksquare$

Example 3.6 Let Ψ be the QCNF shown in chart form on the left of Figure 1. The corresponding Γ_* and Γ_B are shown on the right. Γ_B does not admit a Q-refutation, so we conclude by Lemma 3.5 that Ψ is false. $\square$

It is not practical for a solver to construct $\mathcal{G}_*(\mathcal{F})$ or $\mathcal{G}_B(\mathcal{F})$ explicitly since the sizes of these clause sets might be anywhere from empty to exponentially larger than $\mathcal{F}$. Instead, solvers (ideally) discover clauses in $\mathcal{G}_*(\mathcal{F})$ as the proof search goes along, reduce them to clauses in $\mathcal{G}_B(\mathcal{F})$, and record them. (Some implementations may discover non-minimal consistent hitting sets and not extract a cmhs.) Any partial assignment that satisfies all clauses in $\mathcal{F}$ is a consistent hitting set. If at any point the solver discovers a Q-refutation of whatever guard clauses have been discovered, Ψ is proven to be true.

4 QBF Conflict-Driven Clause-Learning Solvers (Review)

The ***QBF conflict-driven clause-learning*** **(QCDCL)** strategy for PCNF solving is inspired by the great success of conflict-driven clause-learning (CDCL) in propositional SAT solving [11, 12, 2]. Although CDCL is often described in the literature as a variant of DPLL with learning added, it has been argued that the idea of CDCL is actually quite different [11, 16].

Due to universal quantifiers, the QCDCL strategy becomes considerably more technical. To date, the most in-depth treatment in the literature is found in Giunchiglia *et al.* [3], although they call it Q-DLL-LN. This section reviews the main ideas so that QPUP clause learning may be placed in context.

4.1 QCDCL Rounds

A QCDCL solver proceeds by *rounds*. Initially a PCNF $\Psi = \overrightarrow{Q}.\,\mathcal{F}$ is known and $\Gamma_* = \widetilde{Q}.\,\mathcal{G}_*(\mathcal{F})$ (see Section 3) is unknown. As clauses related to $\mathcal{G}_*(\mathcal{F})$ are discovered, they are added to an (initially empty) set of *guard clauses* $\mathcal{G}$. Each round proceeds by *decision levels* until a terminating event occurs. Assignments accumulate throughout a round in a sequence often called the ***trail***, τ. Each assignment is applied to Ψ (and is implicitly applied to Γ_*), giving $\Psi\lceil_\tau$ and $\Gamma_*\lceil_\tau$, and is appended to τ before the next assignment is made. Assignments have categories, such as "assumption," or one of the safe assignments detailed in Section 4.2, to facilitate clause learning after a conflict. The *alevel* (assignment level) of a literal is the decision level at which it was assigned.

At decision-level 0, beginning with an empty τ, *safe* assignments are applied to $\Psi\lceil_\tau$ and $\mathcal{G}\lceil_\tau$, where $\mathcal{G}$ is whatever part of Γ_B has been discovered and recorded. *Safe* assignments are those that cannot change the truth value of $\Psi\lceil_\tau$ and $\Gamma_*\lceil_\tau$. Safe variable assignments continue until no more are found, or until a terminating event occurs.

At positive decision levels, the first variable assignment is an *assumption*, which is *unsafe* (may change the truth value of $\Psi\lceil_\tau$ or $\Gamma_*\lceil_\tau$). For soundness, the assumption literal must not be inner to any unassigned literal.

Subsequent variable assignments on the same decision level are *safe* for $\Psi\lceil_\tau$ and $\Gamma_*\lceil_\tau$. Safe variable assignments continue until no more are found, or until a terminating event occurs. For all decision levels, if no terminating event has occurred, the decision level is increased by 1, a new assumption is made, and the higher decision level continues with additional safe variable assignments, as just described.

4.2 Safe QCDCL Assignments

Perhaps the major complication in going from CDCL to QCDCL is the number of safe assignments to be managed. The first complication is that there are two quite different PCNF formulas being updated and a safe assignment needs to be safe for both formulas. In many cases, it is obviously safe for one formula, but not so clear for the other.

The safe assignments typically implemented in QCDCL-based solvers are:
Unit-clause implication: If $C \in \mathcal{F}$ and $C\lceil_\tau$, followed by universal reductions, contains the single existential literal e, append e to τ.

Guard unit implication: If $C \in \mathcal{G}$ and $C\lceil_\tau$, followed by universal reductions, contains the single existential literal u, based on $\widetilde{Q}$, append u to τ.
Existential pure-literal rule: If the existential literal e appears in some clause in $\mathcal{F}\lceil_\tau$, and $\overline{e}$ does not appear in $\mathcal{F}\lceil_\tau$, then append e to τ.
Universal pure-literal rule: If the universal literal u appears in some clause in $\mathcal{F}\lceil_\tau$, and $\overline{u}$ does not appear in $\mathcal{F}\lceil_\tau$, then append $\overline{u}$ to τ.

4.3 QCDCL Terminating Events

There are three kinds of terminating events for a round:

(1) All clauses in $\mathcal{F}$ are satisfied; i.e., $\mathcal{F}\lceil_\tau$ is empty. Then the assignments in τ comprise a consistent hitting set for $\mathcal{F}$ and a clause of $\mathcal{G}_*(\mathcal{F})$ may be discovered by finding a cmhs and negating it. (In practice an implementation might settle for an over-approximation of the cmhs.) This clause is simplified by universal reductions, keeping in mind that the quantifier types are now dictated by $\widetilde{Q}$, yielding a new guard clause G. If a cmhs was used, $G \in \Gamma_B$, otherwise it is subsumed by some clause in Γ_B. G is added to $\mathcal{G}$. Applying τ falsifies G, since it was a negated hitting set before the universal reductions. Next, assignments are retracted from τ, a complete decision level at a time, from highest to lower, until the remaining assignments no longer falsify G.

(2) A ***conflict*** occurs in some clause $C \in \mathcal{F}$, meaning that τ, coupled with universal reductions in C, have *falsified* C, i.e., $C\lceil_\tau$, followed by universal reductions, is an empty clause. In this case a new clause D is derived from Ψ by Q-resolution and added to $\mathcal{F}$. $D\lceil_\tau$, followed by universal reductions, also is an empty clause. Next, assignments are retracted, a complete decision level at a time, from highest to lower, until the remaining assignments, followed by universal reductions, no longer falsify D.

(3) A conflict occurs in some clause $C \in \mathcal{G}$, meaning that $C\lceil_\tau$, followed by universal reductions, is an empty clause. Note that the only clauses in $\mathcal{G}$ are those added by earlier instances of case 1 and this case. In this case a new guard clause E is derived from $\widetilde{Q}.\,\mathcal{G}$ by Q-resolution[4] and added to $\mathcal{G}$. $E\lceil_\tau$, followed by universal reductions, also is an empty clause. Next, assignments are retracted, a complete decision level at a time, from highest to lower, until the remaining assignments, followed by universal reductions, no longer falsify E.

If D in case 2 is an empty clause, the truth value of Ψ can be proven to be false. If E in case 3 is an empty clause, the truth value of Ψ can be proven to be true. It is straightforward in principle to extract either Q-refutation from a trace of the solver's actions [1, 13]. If G in case 1 is an empty clause, the truth value of Ψ can be shown to be true simply by applying the hitting set underlying G to Ψ, because the hitting set contains only existential variables (based on $\overrightarrow{Q}$). In these cases, the instance is solved.

If no empty clause has been derived, another round is started. The unretracted part of τ from the previous round is the initial value of τ for the next round. Whatever decision levels were not retracted remain in place. Safe assignments continue on the highest unretracted decision level.

[4] Previous descriptions of this general strategy in the literature speak of "cubes," and "term resolution," as separate concepts, but the formulation as a guard formula needs only Q-resolution.

Definition 4.1 An ***asserting clause*** in the context of a trail σ is a clause C such that $C\lceil_{\sigma}$ satisfies the conditions for unit-clause implication in the original formula or for guard unit implication in the guard formula (Section 4.2). I.e., $C\lceil_{\sigma}$ has only one existential literal e (possibly $\bot$), called the ***asserting literal***, and e is not inner to any universal literals in $C\lceil_{\sigma}$. This terminology is primarily used when σ is the trail after backtracking from τ; i.e., σ is a proper prefix of τ. $\square$

An important optimization is that the learned clause, D in case 2 or E in case 3, is asserting after backtracking, so at least one safe assignment is available before a new assumption is needed. When the learned clause is asserting, it is usual to backtrack as many decision levels as possible, while maintaining the asserting property.

By the nature of a round, an asserting learned clause cannot be subsumed by an already known clause, or else it would have become a unit-clause implication or guard unit implication before the completion of the decision level to which the round backtracked after learning this clause. We know this did not happen because at least one higher decision level was started before the asserting clause was learned. Since every round learns a clause, it follows that the number of rounds is finite (as long as all learned clauses are remembered).

5 Learning with QBF Pseudo Unit Propagation (QPUP)

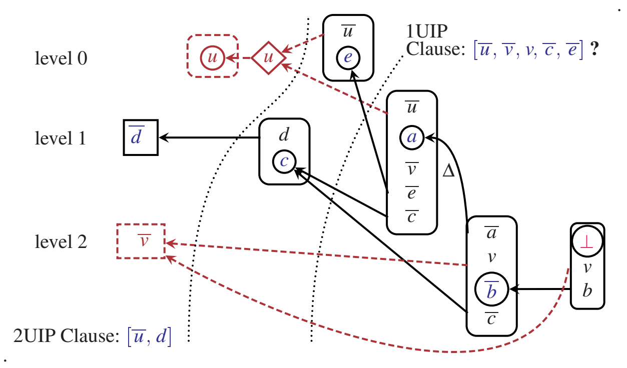
We introduce a practical version of QPUP which can be used to derive a learned clause from a conflict graph. A simple version of QPUP uses the entire conflict graph and was introduced as a theoretical, rather than practical construct [17]. Its point was to show that a non-tautological asserting clause (Definition 4.1) could be learned in time polynomial in the size of the conflict graph (whereas the published methods of learning a non-tautological clause might take exponential time).

This section describes how to selectively apply QPUP after a conflict has occurred, keeping operations confined to recent decision levels as far as practical, in the spirit of the propositional CDCL strategy. This more sophisticated version of QPUP learning is deferred to Section 5.2, until additional nomenclature has been introduced. The procedure is essentially the same for conflicts in the original formula and the guard formula.

5.1 QCDCL Conflict Graphs

During a QCDCL search suppose a falsified clause is encountered after a sequence of assumptions, unit-clause implications, and otherwise-assigned literals, as described in Section 4. Each literal in the trail τ is either an *assumption* or an *implied literal* or an *otherwise-assigned literal* (any other safe assignment).

Definition 5.1 A clause is *effectively unit* in the context of τ if the restriction based on τ, followed by universal reductions (see Definition 2.1), makes the clause a unit clause. A clause is said to be *effectively empty* (or *falsified*) in the context of a partial assignment τ if the restriction based on τ, followed by any applicable universal reductions, makes the clause an empty clause. The ***antecedent*** clause of an implied literal p (denoted *ante*(p)) is the (unique) clause that became effectively unit earliest to imply p. If a clause became effectively empty, we say that $\bot$ is the "implied literal".



level 0
level 1
level 2
1UIP
Clause: $[\overline{u}, \overline{v}, v, \overline{c}, \overline{e}]$?
2UIP Clause: $[\overline{u}, d]$
Δ

Fig. 2. QCDCL conflict graph; see Example 5.2 and later examples. Circles enclose implied literals. Boxes at the left enclose assumptions. Diamonds enclose otherwise-assigned literals. Rounded boxes enclose an antecedent clause. The antecedent clause of a universal implied literal is in the opposite formula and cannot be used for resolution. The rounded box has dashed lines to denote this. Dashed arrows show connections to universal literals. The dotted arcs show various cuts. A solid arrow crossing a cut goes to an existential literal whose negation is definitely in the QCDCL learned clause associated with that cut (cf. propositional CDCL). A dashed arrow always crosses the cut and goes to a universal literal, but its negation may not be in the learned clause.

The *conflict graph* associated with a falsified clause is the rooted directed acyclic graph (DAG) in which $\perp$ is the root vertex and its antecedent is the falsified clause. The remaining vertices are the assumptions, implied literals, and otherwise-assigned literals reachable from $\perp$, based on the directed edges.

The *directed edges* of the conflict graph are (p, q), where p and q are vertices in the conflict graph, p is existential, and $\overline{q} \in ante(p)$. See Figure 2.

Otherwise-assigned literals in the conflict graph for the original formula can arise through the *universal pure-literal rule* and through *guard unit implication*. Like assumptions, otherwise-assigned literals have no antecedent. Existential pure literals are not vertices in the conflict graph of an original formula.

The only differences in a conflict graph for the guard formula are that the vertices with antecedents are guard unit implications (which are existential literals in the context of the guard formula) and the otherwise-assigned literals are *existential pure literals* in the original formula or *unit-clause implications* in the original formula. Both of these literal types are *universal* in the context of the guard formula. Universal pure literals are not in the guard conflict graph. □

Example 5.2 For concreteness we suppose that the QCDCL proof search assumes negative literals. Figure 2 shows a conflict graph for the original formula of Figure 1 that occurs in round 4. In the first three rounds the guard clauses D_1 and D_2 were discovered and the guard clause $[u]$ was derived, causing all decision levels to be retracted.

At the beginning of round 4, u is a guard unit implication at level 0, so it is an otherwise-assigned literal for purposes of the original formula. Now e is implied via C_3. Round 4 continues as follows: Level 1: assume $\overline{d}$; imply c via C_6; imply a via C_2.

Level 2: assume $\overline{v}$; imply $\overline{b}$ via C_7; imply $\bot$ via C_8. Discussion is continued in several subsequent examples. □

Definition 5.3 In a QCNF conflict graph a ***conflict-generating cut*** is a partition of the vertices in the ***reason side*** and the ***conflict side*** such that: **(A)** $\bot$ is on the conflict side; **(B)** every vertex on the conflict side is reachable from $\bot$ by a directed path using only vertices on the conflict side; **(C)** every assumption and otherwise-assigned literal is on the reason side. We abbreviate "conflict-generating cut" to "cut" in later discussions.

A ***unique implication point*** **(UIP)** p is an existential vertex such that all edges from existential literals that are reachable from $\bot$ and are assigned later than p go to: **(D)** p or **(E)** an existential literal assigned later than p or **(F)** an existential literal assigned at a decision level less than p or **(G)** a universal literal. □

In propositional CDCL, the clause associated with a cut consists of the negations of literals on the reason side that are reached by a single edge from some literal on the conflict side. In QCNF this clause might not be derivable in Q-resolution because universal literals can result in tautologous resolvents. Moreover, some universal literals might be removable by universal reduction.

The *UIP cut* for a UIP p in traditional CDCL places all existential literals that are assigned later than p and are in the conflict graph on the conflict side, and places all other literals in the conflict graph on the reason side. This paper refines the definition of the *UIP cut* in the context of QCDCL to permit certain existential literals that are assigned earlier than p to appear on the conflict side. This might be necessary to be able to associate a *non-tautological* clause with the cut.

Example 5.4 Figure 2 shows a typical situation where the traditional UIP cut would derive a tautological clause. The dotted arcs show some conflict-generating cuts. Vertex c is a UIP. The traditional associated 1UIP cut is shown by the rightmost dotted arc. The clause associated with this cut, obtained by resolving the clauses on the conflict side, is tautological. Neither v nor $\overline{v}$ can be eliminated by universal reduction using only clauses on the conflict side because they are always blocked by b or $\overline{c}$ or $\overline{e}$.

Long distance resolution was proposed by Zhang and Malik [19] to accommodate such tautological clauses. Giunchiglia and co-authors pioneered the derivation of learned clauses using Q-resolution and avoiding tautological clauses [3]. □

5.2 QPUP Clauses

The idea of QPUP is that resolutions are performed that mimic the derivation of the implied literal by unit-clause resolution, treating certain earlier-assigned existential literals as unit clauses. However, the *negations* of those earlier-assigned literals are included in the QPUP clause. In this sense, QPUP extends propositional Pseudo Unit Propagation (PUP) [18].

Definition 5.5 Let a conflict graph and a conflict-generating cut be given, as described in Section 5.1. Recall *alevel* (assignment level) from Section 4.1. Each assumption increases the *alevel* by one and subsequently assigned literals up to the next assumption take this value for their *alevel*. Let *dlevel* be the decision level at which the conflict occurred.

A function *qpup*$(e,\, a\ell)$ is any function that returns a clause for each existential literal e on the conflict side of the cut, including $\bot$, with certain properties:

1. If all edges leaving e go to the reason side, then *qpup*$(e,\, a\ell) =$ *ante*(e).
2. If $\overline{r} \in$ *qpup*$(e,\, a\ell)$, then r is on the reason side and is reachable by a single edge from some vertex on the conflict side.
3. Let A be the set of existential literals in *qpup*$(e,\, a\ell)$ with *alevel* $\leq a\ell$. If $\overline{r} \in$ *qpup*$(e,\, a\ell)$ and r is universal and r would be unassigned after retracting all assignments with *alevel* $> a\ell$, then r is tailing in $($*qpup*$(e,\, a\ell) - A - e)$.
4. A clause that subsumes *qpup*$(e,\, a\ell)$ can be derived by Q-resolution from *ante*(e) and the clauses *qpup*$(r_i,\, a\ell)$ such that $\overline{r_i} \in$ *ante*(e) and r_i is on the conflict side. (Note that a clause subsumes itself.) □

The properties are well defined because the conflict graph is acyclic. Note that properties 1 and 4 ensure that *qpup*$(e,\, a\ell)$ is not tautological.

Not every cut permits *qpup* to be defined. Suppose *qpup* can be defined for a given cut and a backtrack level $a\ell <$ *dlevel*. The important invariant for $D =$ *qpup*$(e,\, a\ell)$ is

$$\mathsf{unrd}_*(\textit{qpup}(e,\, a\ell) - \{e\}) \subseteq \textit{qpup}(\bot,\, a\ell) \qquad (2)$$

I.e., if e is removed from D and all possible universal reductions are performed in $D - e$, then the remaining literals are in *qpup*$(\bot,\, a\ell)$. Note that the invariant holds also when $e = \bot$. Given a conflict graph, the goal is to identify a suitable conflict-generating cut such that the clause *qpup*$(\bot,\, a\ell)$ is non-tautological and asserting and hence can be used as a learned clause.

Example 5.6 Referring again to Figure 2, it is easy to see by inspection that *qpup* cannot be defined for the rightmost cut for any $a\ell <$ *dlevel*, because a is on the conflict side and $\overline{a}$ cannot be resolved out without creating a tautology.

For the leftmost cut, *qpup* is easily defined for $a\ell = 0$. For c and e their *qpup* is their antecedent. Then *qpup*$(a,\, 0) =$ *ante*$(a) *_e$ *qpup*$(e,\, 0) *_c$ *qpup*$(c,\, 0)\, \Delta_{\overline{v}} = [\overline{u},\, d,\, a]$. This can be resolved against the clause with $\overline{a}$, etc., without creating a tautology. Finally, *qpup*$(\bot,\, 0) = [\overline{u},\, d]$, which is asserting for level 0.

Note that *alevel*$(e) <$ *alevel*$(\overline{d})$, but including e on the conflict side is necessary to enable tautologies to be avoided. Finally, it is worth noting that traditional QCDCL based on [3] derives the weaker clause $[\overline{u},\, d,\, \overline{v},\, \overline{e}]$ by starting at the falsified clause $[v, b]$ and resolving on b, c, a, and again on c. □

Our principal contribution is the description (in the next section) and experimental evaluation of a practical procedure to identify a cut and a value of $a\ell$ such that *qpup* is efficient to compute and has the further property that *qpup*$(\bot,\, a\ell)$ is non-tautological and asserting after backtracking.

5.3 Cuts and Backtrack Levels for QPUP

This section provides an abstract description of our procedure to identify a suitable cut and backtrack level for QPUP after a conflict. For simplicity a conflict in the original formula is assumed.

The key idea is that an ***agenda*** is constructed and processed. The agenda is a sequence of literals in trail-assignment order that are relevant to deriving the needed $qpup$ clauses. As an exception, unassigned universal literals appear with the same clause that caused them to be inserted. Processing the agenda corresponds to finding a suitable cut in iterative fashion. The procedure is illustrated by Examples 5.7 and 5.8 to make it easier to follow the general description.

Starting from an empty cut, existential literals are put on the conflict side until a UIP cut is found. If that cut cannot be associated with a non-tautological clause, processing continues and further literals are put on the conflict side, thus modifying the cut. During processing, no clauses are constructed explicitly. The state of the agenda is inspected to check if the clause associated with the current cut is non-tautological and asserting.

Literals are annotated with some status information. We use these graphical mnemonics: p/k denotes that *alevel*$(p) = k$ (∞ denotes it is unassigned); $\textcircled{e}$ means that e must be on the conflict side and its *qpup* must eventually be computed to produce a Q-derivation of the learned clause; $\boxed{q}$ denotes that q is a suitable UIP, i.e., *qpup* will be computed with $a\ell = ($*alevel*$(q) - 1)$ and $\overline{q}$ will be the asserting literal in *qpup*$(\bot,\, a\ell)$, the learned clause.

Literals are processed from right to left, i.e., in reverse trail order. Universal literals are skipped over and each existential literal is processed just once. Processing may cause other literals to be inserted into the agenda, but such inserts are always to the left of the literal being processed, i.e., in the part of the agenda yet to be processed.

In general terms, the processing has two phases: In the first phase a suitable UIP literal is identified and this establishes the value of $a\ell$ for which *qpup* is needed. In the second phase the complete cut is identified.

The agenda is initialized with the literals in *ante*$(\bot)$, with $\textcircled{\bot}$ rightmost, unassigned universals immediately to its left, and remaining literals further to the left, in trail order. For phase 1, proceeding right to left, examine the next existential literal, say $\overline{e}/k$.

(1) If some literal to the left of $\overline{e}$ has *alevel* $= k$, e must go on the conflict side. **(2)** If e is inner to some pair of complementary universal literals in the agenda, e must go on the conflict side. **(3)** If e is inner to some universal literal u/m in the agenda where $m \geq k$, e must go on the conflict side.

If none of the above tests determines that e must go on the conflict side, then change the notation to $\overline{\mathbf{e}}/k$, denoting that $\overline{e}$ will be the asserting literal, set $a\ell = k - 1$, and proceed to phase 2.

Otherwise, replace $\overline{e}/k$ with *ante*(e) in the agenda, as follows: $\textcircled{e}$ replaces $\overline{e}/k$; unassigned universal literals that are not already in the agenda are inserted immediately to the left of $\textcircled{e}$; remaining literals of *ante*(e) that are not already in the agenda are inserted in correct trail order (somewhere to the left of $\textcircled{e}$), annotated with their *alevel*s. Continue phase 1 to the left of $\textcircled{e}$.

If phase 1 reaches the left end of the agenda without identifying an asserting literal, then *qpup*$(\bot,\, 0)$ will reduce to the empty clause.

Example 5.7 Referring again to Figure 2, Table 1 shows the evolution of the agenda through phase 1. After step 1, $\overline{a}$ cannot be the asserting literal because $\overline{c}$ is earlier on the same *alevel*. After step 2, $\overline{c}$ cannot be the asserting literal because it is inner to v

Table 1. Agenda processing discussed in Examples 5.7 and 5.8.

Step	Agenda								
0							$v/2$	$b/2$	$\textcircled{\bot}$
1					$\overline{c}/1$	$\overline{a}/1$	$v/2$	$\textcircled{\overline{b}}$	$\textcircled{\bot}$
2		$\overline{u}/0$	$\overline{e}/0$	$\overline{c}/1$	$\overline{v}/2$	$\textcircled{a}$	$v/2$	$\textcircled{\overline{b}}$	$\textcircled{\bot}$
3	$\overline{u}/0$	$\overline{e}/0$	$d/1$	$\textcircled{c}$	$\overline{v}/2$	$\textcircled{a}$	$v/2$	$\textcircled{\overline{b}}$	$\textcircled{\bot}$
4	$\overline{u}/0$	$\overline{e}/0$	$\boxed{d}/1$	$\textcircled{c}$	$\overline{v}/2$	$\textcircled{a}$	$v/2$	$\textcircled{\overline{b}}$	$\textcircled{\bot}$
					end phase 1				
5	$\overline{u}/0$	$\textcircled{e}$	$\boxed{d}/1$	$\textcircled{c}$	$\overline{v}/2$	$\textcircled{a}$	$v/2$	$\textcircled{\overline{b}}$	$\textcircled{\bot}$

and $\overline{v}$. After step 3, d can be the asserting literal because it is *not* inner to v and $\overline{v}$. Step 4 terminates phase 1. □

Phase 2 begins immediately to the left of the asserting literal in the agenda. Proceeding right to left as in phase 1, examine the next existential literal, say $\overline{f}/j$. We know $j \leq a\ell$. If f is inner to some pair of complementary universal literals in the agenda, f must go on the conflict side. Otherwise, $\overline{f}/j$ remains unchanged in the agenda and processing moves left. If f must go on the conflict side, replace $\overline{f}/j$ with *ante*(f) in the agenda, following the same procedure as above for replacing $\overline{e}/k$ with *ante*(e). In particular $\textcircled{f}$ now appears in the agenda and phase 2 continues to the left of $\textcircled{f}$. Phase 2 terminates after the leftmost literal in the agenda has been processed. At this point, the UIP cut has been found and is associated with the non-tautological asserting clause *qpup*$(\bot,\, a\ell)$, which is to be learned.

Example 5.8 Continuing from Example 5.7, the end of Table 1 shows the evolution of the agenda through phase 2. After step 4, $\overline{e}$ cannot be on the reason side because it is inner to v and $\overline{v}$, so it is replaced by *ante*(e). After step 5, no existential literals inner to v and $\overline{v}$ remain on the reason side and processing terminates. Literal d is the UIP. Literals e, c, a and $\overline{b}$ are on the conflict side. □

After termination of agenda processing, clauses *qpup*$(e,\, a\ell)$ are computed for every existential literal e on the conflict side. The computations are done *in* trail order, i.e., from left to right in the agenda. The order is important because the computation of some *qpup*$(e_2,\, a\ell)$ might use *qpup*$(e_1,\, a\ell)$ if e_1 is left of e_2 in the agenda, i.e., earlier on trail. Finally, the clause *qpup*$(\bot,\, a\ell)$ is computed. Example 5.6 illustrates the computation of the QPUP clauses referring to the agenda from Example 5.8.

Lemma 5.9 The agenda-based procedure presented above **(A)** has worst-case run time which is polynomial in the size of the conflict graph and **(B)** allows to derive a non-tautological asserting learned clause $C =$ *qpup*$(\bot,\, a\ell)$. ■

5.4 Lazy QPUP

The agenda-based approach from the previous section allows for a lazy form of QPUP learning where *no* resolutions are carried out. In effect, it inspects the final agenda to determine which literals will be in *qpup*$(\bot,\, a\ell)$, i.e., the learned clause. No literals on the conflict side (those enclosed in circles) will appear; all existential literals on the

Table 2. Running times in seconds on *qdpllexp* family. "segv" denotes "segmentation violation".

family index	18	19	20	21	22	23
`QuBE 1.3`	10	22	47	105	segv	segv
`DepQBF 0.1`	8	16	32	69	140	298
`CirQit 3.15`	1	1	3	5	11	21
`DepQBF QPUP`	.00	.00	.00	.00	.00	.00

Note: `DepQBF QPUP` does not register any CPU time, even up to level 99. The run logs show that the resolution count increases by 8 for each level in the family.

reason side *must* appear. For universal literals, those that can be reduced out at the end and complementary pairs definitely will not appear. To be safe, other universal literals are kept, but they cannot prevent the learned clause from being asserting after backtracking to $a\ell$.

6 Experimental Results

We implemented a prototype version of QPUP learning and lazy QPUP as described in Sections 5.2 and 5.4 in the open-source search-based QBF solver `DepQBF`[5] for comparisons with traditional QCDCL. QPUP learning is applied to the original formula as well as to the guard formula. It is compatible with all the sophisticated techniques already in `DepQBF`, including pure literal detection, proof generation for certificate extraction [1, 13], and the standard dependency scheme.

It has been reported that QCDCL clause learning, published as *Q-DLL-LN* [3] and used in other solvers (often under the label *QDPLL*), can spend time that is exponential in the size of the conflict graph to learn a single clause [17]. A family of small instances was given that elicits exponential behavior. Since the original motivation for QPUP was to avoid this behavior, we checked it on this family. The first three lines of Table 2, reproduced from [17], provide empirical confirmation of the theoretical analysis that the running time for traditional QDPLL learning doubles for each increase of one level in the family. The note shows that QPUP does not experience exponential growth.

To compare `DepQBF` with QPUP learning, lazy QPUP and traditional QCDCL, we considered the 276 preprocessed instances used for *QBFEVAL-12 Second Round* (QBFEVAL-12-SR). We did not apply any further preprocessing to these instances.[6] For additional tests we used preprocessed instances from QBFEVAL-10 which were not solved by preprocessing. We ran experiments on 64-bit Linux AMD Opteron 6176 SE with a time limit of 900 seconds and a memory limit of 7 GB.[7]

Table 3 shows a comparison of `DepQBF` with traditional QCDCL, QPUP learning and lazy QPUP. Figure 3 shows these results in a cactus plot. Lazy QPUP solves the largest number of instances, both overall and individually with respect to satisfiable and unsatisfiable ones. The PAR10 time (i.e., average time with timeouts multiplied by 10) of lazy QPUP is moderately smaller than the time of the other two configurations.

[5] Please visit http://lonsing.github.com/depqbf/ for released versions. The version reported in this section is available from the first author.

[6] Visit http://fmv.jku.at/seidl/qbfeval2012r2/ and select "eval12bloqqer,"

[7] Please visit http://www.kr.tuwien.ac.at/staff/lonsing/sat13submission.tar.gz for binaries, logs and a longer report.

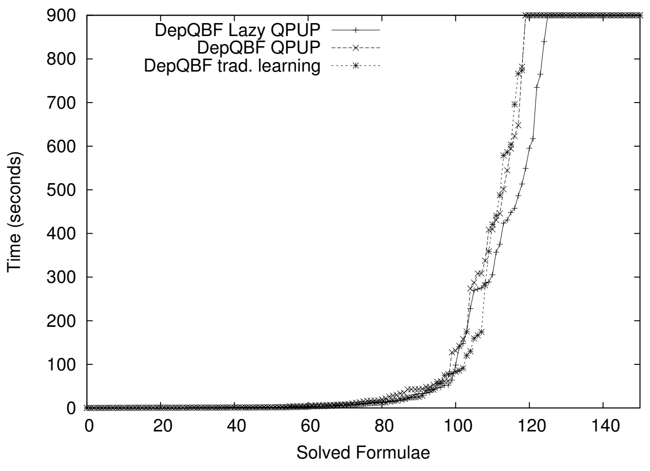


Fig. 3. Cactus plot of run times related to Table 3 for QBFEVAL-12-SR.

Table 4 shows detailed statistics on instances solved by both QPUP learning and traditional QCDCL. Due to the large overlap in solved instances these statistics are comparable. (Lazy QPUP counts closely match QPUP.) QPUP is higher on most counts, but is 12% lower on backtracks. Additional tests on the 373 preprocessed QBFEVAL-10 instances that were not solved during preprocessing showed a similar pattern, and are omitted to save space.

When all three methods are forced to use the same variable re-weighting policy, it becomes clear that QPUP produces shorter learned clauses than traditional QCDCL. It is noteworthy that the ***logical computation*** (meaning the sequence of assumptions, safe assignments, learned clauses and backtracks) was exactly the same on 228 of the 273 instances solved by both methods on preprocessed QBFEVAL-10 instances. This confirms that QPUP almost always produces the same learned constraint as traditional QCDCL. For the 45 instances that did differ in their logical computations Table 5 shows some statistics. In one striking instance,[8] both methods learned the same first four clauses, but then the fifth learned clause by traditional QCDCL was a superset of that learned by QPUP, and contained 128 additional literals (534 vs. 406).

Lazy QPUP performed the same logical computation as QPUP on 97% of the 273 instances reported in Table 5 and used 111 CPU Seconds on average. In addition it solved one additional instance on which QPUP timed out, which we attribute to a faster procedure, not a different learning strategy.

The overall picture in Table 3, the reduced numbers of backtracks in Tables 4 and 5, and the reduced learned-clause lengths in Table 5 show the potential of QPUP learning. Higher resolution counts for QPUP are expected when QCDCL does not encounter conflicting universal literals, based on the analysis of PUP [18], but other benefits of QPUP appear to compensate.

[8] `TOILET16.1.iv.32-shuffled`

Table 3. (Left) `DepQBF` with QPUP, lazy QPUP, and traditional QCDCL on the 276 preprocessed instances from QBFEVAL-12-SR. Times are average including timeouts multiplied by 10 (PAR10).

Table 4. (Right) Comparison of QPUP learning and traditional QCDCL based on the runs from Table 3. Averages are based on 113 instances solved by both configurations. "Resolutions" and "length" are per learned clause.

	Solved (sat,unsat)	Time (avg.)
`Trad. QCDCL`	119 (62, 57)	5,148
`QPUP`	119 (63, 56)	5,151
`Lazy QPUP`	125 (65, 60)	4,963

Both Solved	Trad. vs. QPUP	
Time	55.21	51.93
Assignments	$9.1{\cdot}10^6$	$11.1{\cdot}10^6$
Backtracks	59,000	52,000
Resolutions	23.50	34.05
Length	53.58	82.50

Table 5. Comparison of traditional QCDCL and QPUP learning on the 45 instances with differing logical computation among the 273 preprocessed QBFEVAL-10 instances solved by both configurations. Statistics are average values and standard deviation (σ) of the difference.

	Trad. QCDCL	QPUP	Difference	σ of Diff.
No. Learned Clauses	228,666	154,379	74,287	210,332
Learned Clause Length	191.8	131.4	60.4	141.7
No. Clause Resolutions	$3.96{\cdot}10^6$	$4.80{\cdot}10^6$	$-0.84{\cdot}10^6$	$4.93{\cdot}10^6$
No. Learned Cubes	79,555	89,723	-10,168	54,825
Learned Cube Length	409.2	408.3	0.9	7.2
No. Cube Resolutions	51,674	107,167	-55,493	237,015
Backtracks	268,528	209,942	58,586	216,325
CPU Seconds	208	186	22	111

7 Conclusion

This paper presented *QPUP learning*, a novel approach to conflict-driven clause-learning (QCDCL) in QBF solvers. Given a conflict graph, the idea is to resolve on variables in the *same* order as they were assigned, rather than in reverse order. In contrast to traditional QCDCL, QPUP learning is a polynomial-time procedure.

The implementation of QPUP learning in `DepQBF` is compatible with sophisticated techniques like pure literals, dependency schemes [14, 15, 8], proof generation and hence also certificate extraction. Experimental results show the potential of QPUP learning but, at the same time, identified several procedural optimizations as future work.

Further research directions are comparison of certificates obtained by resolution proofs based on traditional QCDCL and QPUP learning, a detailed analysis of the learned clauses, and formally proving the correctness of QPUP learning in the context of guard formulas and dependency schemes, building upon the framework of *guard formulas*.

References

1. V. Balabanov and J. R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41:45–65, 2012.
2. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT (LNCS 2919)*, pages 502–518. Springer, 2004.
3. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *JAIR*, 26:371–416, 2006.
4. A. Goultiaeva and F. Bacchus. Exploiting QBF duality on a circuit representation. In *AAAI*, 2010.
5. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117:12–18, 1995.
6. H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, 1999.
7. W. Klieber, S. Sapra, S. Gao, and E. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *SAT, LNCS*, 2010.
8. F. Lonsing and A. Biere. A compact representation for syntactic dependencies in QBFs. In *SAT*, pages 398–411. Springer, 2009.
9. F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver: System description. *JSAT*, 7:71–76, 2010.
10. F. Lonsing and A. Biere. Integrating dependency schemes in search-based QBF solvers. In *SAT*, pages 158–171. Springer, 2010.
11. J. P. Marques-Silva and K. A. Sakallah. GRASP–a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.
12. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, June 2001.
13. A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere. Resolution-based certificate extraction for QBF (tool presentation). In *Proc. SAT*, 2012.
14. M. Samer. Variable dependencies of quantified CSPs. In *LPAR, LNCS 5330*, pages 512–527, 2008.
15. M. Samer and S. Szeider. Backdoor sets of quantified boolean formulas. *JAR*, 42:77–97, 2009.
16. A. Van Gelder. Generalized conflict-clause strengthening for satisfiability solvers. In *Proc. SAT (LNCS 6695)*, pages 329–342. Springer, 2011.
17. A. Van Gelder. Contributions to the theory of practical quantified boolean formula solving. In *Proc. CP*, 2012.
18. A. Van Gelder. Producing and verifying extremely large propositional refutations: Have your cake and eat it too. *AMAI*, 65(4):329–372, 2012.
19. L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proc. ICCAD*, pages 442–449, 2002.