Journal of Artificial Intelligence Research 65 (2019) 181-208 Submitted 10/2018; published 06/2019

Dependency Learning for QBF

Tomáš Peitl PEITL@AC.TUWIEN.AC.AT
Friedrich Slivovsky FS@AC.TUWIEN.AC.AT
Stefan Szeider SZ@AC.TUWIEN.AC.AT
Algorithms and Complexity Group, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria

Abstract

Quantified Boolean Formulas (QBFs) can be used to succinctly encode problems from domains such as formal verification, planning, and synthesis. One of the main approaches to QBF solving is Quantified Conflict Driven Clause Learning (QCDCL). By default, QCDCL assigns variables in the order of their appearance in the quantifier prefix so as to account for dependencies among variables. Dependency schemes can be used to relax this restriction and exploit independence among variables in certain cases, but only at the cost of nontrivial interferences with the proof system underlying QCDCL. We introduce *dependency learning*, a new technique for exploiting variable independence within QCDCL that allows solvers to learn variable dependencies on the fly. The resulting version of QCDCL enjoys improved propagation and increased flexibility in choosing variables for branching while retaining ordinary (long-distance) Q-resolution as its underlying proof system. We show that dependency learning can achieve exponential speedups over ordinary QCDCL. Experiments on standard benchmark sets demonstrate the effectiveness of this technique.

1. Introduction

Conflict Driven Clause Learning (CDCL) represents the state of the art in propositional satisfiability (SAT) solving (see, e.g., the work of Marques-Silva, Lynce, & Malik, 2009). Modern CDCL solvers can handle input formulas with thousands of variables and millions of clauses (Malik & Zhang, 2009). Their remarkable performance has led to the adoption of SAT solving in electronic design automation (see the survey of Vizel, Weissenbacher, & Malik, 2015), it has turned algorithms relying on SAT oracles into viable tools for solving hard problems (see, e.g., the work of Meel et al., 2016), and it has even helped resolve open questions in combinatorics (Heule, Kullmann, & Marek, 2016).

Encouraged by this success, researchers are turning to an even harder problem: the satisfiability problem of Quantified Boolean Formulas (QSAT). Quantified Boolean Formulas (QBFs) enrich propositional formulas with universal and existential quantification over truth values and offer much more succinct encodings of problems from domains such as planning and synthesis (Faymonville et al., 2017). This expressive power comes at a price: QSAT is complete for the complexity class PSPACE and thus believed to be significantly harder than SAT.

Quantified CDCL (Cadoli et al., 2002; Zhang & Malik, 2002) is a natural generalization of CDCL and one of the two[1] dominant algorithmic paradigms in QSAT solving. While the

1. The other paradigm can be broadly characterized by the use of quantifier expansion (or *abstraction*) in combination with SAT oracles (Biere, 2004; Lonsing & Biere, 2008; Janota, Klieber, Marques-Silva, &

performance of QCDCL solvers has much improved over the past years, they have so far failed to replicate the success of CDCL in the domain of SAT.

One of the main obstacles in lifting CDCL to QSAT is that the alternation of existential and universal quantifiers in the quantifier prefix of a QBF (we consider formulas in prenex normal form) introduces dependencies among variables that have to be respected by the order of variable assignments. The resulting constraints not only reduce the empirical effectiveness of branching heuristics but impose severe theoretical limits on the power of QCDCL (Janota, 2016). By default, QCDCL only considers variables from the outermost quantifier block with unassigned variables for branching. In the worst case, this forces solvers into a fixed branching order. Among several techniques that have been introduced to relax this restriction, *dependency schemes* are arguably the most general. Given a QBF, a dependency scheme efficiently computes an overapproximation of its variable dependencies—that is, the result contains every pair of variables for which there is a "real" dependency, but it may contain "spurious" dependencies. Lonsing and Biere (2010) introduced a generalization of QCDCL that uses dependency schemes to relax constraints on the order of variable assignments and implemented this algorithm in the solver DepQBF.

The use of dependency schemes within DepQBF often leads to performance improvements, but it has several drawbacks. First, it changes the proof system underlying constraint learning. Proving soundness of the resulting algorithm is nontrivial even for a simple version of QCDCL and common dependency schemes (Slivovsky & Szeider, 2016; Peitl, Slivovsky, & Szeider, 2016). The continuous addition of solver features makes QCDCL a moving target, and the integration of dependency schemes with any new technique usually requires a new soundness proof. Second, even if soundness of the resulting proof system can be established, efficient (linear-time) strategy extraction from proofs—a common requirement for applications—does not follow. Third, and perhaps most importantly, the syntactic criteria for identifying dependencies used by common dependency schemes (such as the standard dependency scheme or the resolution-path dependency scheme) are fairly coarse, so that the set of dependencies computed by such schemes frequently coincides with the "trivial" dependencies implicit in the quantifier prefix (see Table 3 in Section 5).

In this paper, we describe a new approach to exploiting variable independence in QCDCL solvers we call *dependency learning*. The idea is that the solver maintains a set D of dependencies that is used in propagation and choosing variables for branching just like in QCDCL with dependency schemes: a clause is considered unit under the current assignment if it contains a single existential variable that does not depend, according to D, on any universal variable remaining in the clause; a variable is eligible for branching if it does not depend, according to D, on any variable that is currently unassigned (cf. Biere & Lonsing, 2010). But instead of initializing D using a dependency scheme, dependencies are added on the fly as needed. Initially, the set D is empty, so every clause containing a single existential variable is considered unit and variables can be assigned in any order. When propagation runs into a conflict, the solver attempts to derive a new clause by long-distance Q-resolution (Balabanov & Jiang, 2012; Egly, Lonsing, & Widl, 2013). Because propagation implicitly performs universal reduction relative to D, but Q-resolution applies universal reduction according to the prefix order, the solver may be unable to generate a learned clause. Whenever such a

Clarke, 2012; Rabe & Tentrup, 2015; Janota & Marques-Silva, 2015; Tentrup, 2016; Scholl & Pigorsch, 2016).

situation arises during learning, a set of missing variable dependencies is identified as the (local) cause and added to D. The resulting version of QCDCL potentially improves on the flexibility afforded by dependency schemes but retains long-distance Q-resolution as its underlying proof system and thus enjoys linear-time strategy extraction (Balabanov et al., 2015).

To explore the effectiveness of this technique, we implemented Qute, a QCDCL solver that supports dependency learning. In experiments with benchmark instances from the 2016–2018 QBF Evaluations, Qute is highly competitive with state-of-the-art QBF solvers on both formulas in prenex conjunctive normal form (PCNF), as well as non-CNF formulas in the QCIR format. A comparison of various configurations shows that dependency learning has a significant positive impact on Qute's performance.

Additional experiments show that the number of dependencies learned by Qute on PCNF instances preprocessed by HQSPre (Wimmer et al., 2017) is typically only a fraction of those identified by the standard dependency scheme and even the (reflexive) resolution-path dependency scheme, and that Qute can deal with formulas that are provably hard to solve for vanilla QCDCL (Janota, 2016). We explain the latter result by formally proving that QCDCL with dependency schemes can solve these formulas in polynomial time.

The remainder of this paper is organized as follows. In Section 2, we cover basic definitions and notation. In Section 3, we introduce QCDCL and (long-distance) Q-resolution, its underlying proof system. In Section 4, we describe how to modify QCDCL to support dependency learning, and prove that the resulting algorithm is sound and terminating. In Section 5, we provide an experimental evaluation of Qute. In Section 6, we prove that dependency learning can achieve an exponential speedup over ordinary QCDCL. In Section 7, we study some theoretical properties of dependencies (not) learned by a dependency-learning solver, and in Section 8, we conclude with a discussion of our results and future research directions.

2. Preliminaries

We consider QBFs in Prenex Conjunctive Normal Form (PCNF), i.e., formulas $\Phi = \mathcal{Q}.\varphi$ consisting of a (quantifier) prefix $\mathcal{Q}$ and a propositional CNF formula φ, called the *matrix* of Φ. The *prefix* is a sequence $\mathcal{Q} = Q_1 x_1 \dots Q_n x_n$, where $Q_i \in \{\forall, \exists\}$ is a universal or existential quantifier and the x_i are variables. We write $x_i \prec_\Phi x_j$ if $1 \leq i < j \leq n$ and $Q_i \neq Q_j$, dropping the subscript if the formula Φ is understood. A *CNF formula* is a finite conjunction $C_1 \wedge \dots \wedge C_m$ of clauses, a *clause* is a finite disjunction $(\ell_1 \vee \dots \vee \ell_k)$ of literals, and a *literal* is a variable x or a negated variable $\neg x$. Dually, a *DNF formula* is a finite disjunction $T_1 \vee \dots \vee T_k$ of terms, and a *term* is a finite conjunction $(\ell_1 \wedge \dots \wedge \ell_k)$ of literals. Whenever convenient, we consider clauses and terms as sets of literals, CNF formulas as sets of clauses, and DNF formulas as sets of terms. We assume that PCNF formulas are *closed*, so that every variable occurring in the matrix appears in the prefix, and that each variable appearing in the prefix occurs in the matrix. We write $var(x) = var(\neg x) = x$ to denote the variable associated with a literal and let $var(C) = \{\, var(\ell) : \ell \in C \,\}$ if C is a clause, $var(\varphi) = \bigcup_{C \in \varphi} var(C)$ if φ is a CNF formula, and $var(\Phi) = var(\varphi)$ if $\Phi = Q.\varphi$ is a PCNF formula.

$$\frac{C_1 \vee e \qquad \neg e \vee C_2}{C_1 \vee C_2}\ \text{(resolution)}$$

The *resolution* rule allows the derivation of $C_1 \vee C_2$ from clauses $C_1 \vee e$ and $\neg e \vee C_2$, provided that the *pivot* variable e is existential and that $e \prec \mathit{var}(\ell_u)$ for each universal literal $\ell_u \in C_1$ such that $\overline{\ell_u} \in C_2$. The clause $C_1 \vee C_2$ is called the resolvent of $C_1 \vee e$ and $\neg e \vee C_2$.

$$\frac{C}{C \setminus \{u, \neg u\}}\ \text{(universal reduction)}$$

The *universal reduction* rule admits the deletion of a universal variable u from a clause C under the condition that $e \prec u$ for each existential variable e in C.

Figure 1: Long-distance Q-resolution.

An *assignment* is a sequence $\sigma = (\ell_1, \ldots, \ell_k)$ of literals such that $\mathit{var}(\ell_i) \neq \mathit{var}(\ell_j)$ for $1 \leq i < j \leq n$. If S is a clause or term, we write $S[\sigma]$ for the the result of applying σ to S. For a clause C, we define $C[\sigma] = \top$ if $\ell_i \in C$ for some $1 \leq i \leq k$, and $C[\sigma] = C \setminus \{\overline{\ell_1}, \ldots, \overline{\ell_k}\}$ otherwise, where $\overline{x} = \neg x$ and $\overline{\neg x} = x$. For a term T, we let $T[\sigma] = \bot$ if $\overline{\ell_i} \in T$ for some $1 \leq i \leq k$, and $T[\sigma] = T \setminus \{\ell_1, \ldots, \ell_k\}$ otherwise. An assignment σ *falsifies* a clause C if $C[\sigma] = \emptyset$; it *satisfies* a term T if $T[\sigma] = \emptyset$. For CNF formulas φ, we let $\varphi[\sigma] = \{\, C[\sigma] : C \in \varphi, C[\sigma] \neq \top \,\}$, and for PCNF formulas $\Phi = Q.\varphi$, we let $\Phi[\sigma] = Q'.\varphi[\sigma]$, where Q' is obtained by deleting variables from Q not occurring in $\varphi[\sigma]$.

The semantics of a PCNF formula Φ is defined as follows. If Φ does not contain any variables, then Φ is true if its matrix is empty and false if its matrix contains the empty clause $\emptyset$. Otherwise, let $\Phi = Qx\mathcal{Q}.\varphi$. If $Q = \exists$ then Φ is true if $\Phi[(x)]$ is true or $\Phi[(\neg x)]$ is true, and if $Q = \forall$ then Φ is true if both $\Phi[(x)]$ and $\Phi[(\neg x)]$ are true.

3. QCDCL and Q-resolution

We briefly review QCDCL and Q-resolution (Kleine Büning, Karpinski, & Flögel, 1995), its underlying proof system. More specifically, we consider *long-distance Q-resolution*, a version of Q-resolution that admits the derivation of tautological clauses in certain cases. Although this proof system was already used in early QCDCL solvers (Zhang & Malik, 2002), the formal definition shown in Figure 1 was given more recently (Balabanov & Jiang, 2012). A dual proof system called *(long-distance) Q-consensus*, which operates on terms instead of clauses, is obtained by swapping the roles of existential and universal variables (the analog of universal reduction for terms is called *existential reduction*).

A (long-distance) Q-resolution *derivation* from a PCNF formula Φ is a sequence of clauses such that each clause appears in the matrix of Φ or can be derived from clauses appearing earlier in the sequence using resolution or universal reduction. A derivation of the empty clause is called a *refutation*, and one can show that a PCNF formula is false, if, and only if, it has a long-distance Q-resolution refutation (Balabanov & Jiang, 2012). Dually, a PCNF formula is true, if, and only if, the empty term can be derived from a DNF representation of its matrix by Q-consensus, the dual of Q-resolution.

Starting from an input PCNF formula, QCDCL generates ("learns") *constraints*—clauses and terms—until it produces an empty constraint. Every clause learned by QCDCL can be derived from the input formula by Q-resolution, and every term learned by QCDCL can be derived by Q-consensus (Giunchiglia, Narizzano, & Tacchella, 2006; Egly et al., 2013). Accordingly, the solver outputs TRUE if the empty term is learned, and FALSE if the empty clause is learned.

One can think of QCDCL solving as proceeding in rounds. Along with a set of clauses and terms, the solver maintains an assignment σ (the *trail*). During each round, this assignment is extended by quantified Boolean constraint propagation (QBCP) and—possibly—branching.

Quantified Boolean constraint propagation consists of exhaustive application of universal and existential reduction in combination with unit assignments.[2] More specifically, QBCP reports a clause C as falsified if $C[\sigma] \neq \top$ and universal reduction can be applied to $C[\sigma]$ to obtain the empty clause. Dually, a term T is considered satisfied if $T[\sigma] \neq \bot$ and $T[\sigma]$ can be reduced to the empty term. A clause C is *unit* under σ if $C[\sigma] \neq \top$ and universal reduction yields the clause $C' = (\ell)$, for some existential literal ℓ such that $var(\ell)$ is unassigned. Dually, a term T is *unit* under σ if $T[\sigma] \neq \bot$ and existential reduction can be applied to obtain a term $T' = (\ell)$ containing a single universal literal ℓ. If $C = (\ell)$ is a unit clause, then the assignment σ has to be extended by ℓ in order not to falsify C, and if $T = (\ell)$ is a unit term, then σ has to be extended by $\overline{\ell}$ in order not to satisfy T. If several clauses or terms are unit under σ, QBCP nondeterministically picks one and extends the assignment accordingly. This is repeated until a constraint is empty or no unit constraints remain.

If QBCP does not lead to an empty constraint, the assignment σ is extended by *branching*. That is, the solver chooses an unassigned variable x such that every variable y with $y \prec x$ is assigned, and extends the assignment σ by x or $\neg x$.

The resulting assignment can be partitioned into so-called *decision levels*. The decision level of an assignment σ is the number of literals in σ that were assigned by branching. The decision level of a literal ℓ in σ is the decision level of the prefix of σ that ends with ℓ. Note that each decision level greater than 0 can be associated with a unique variable assigned by branching.

Eventually, the assignment maintained by QCDCL must falsify a clause or satisfy a term. When this happens (this is called a *conflict*), the solver proceeds to *conflict analysis* to derive a learned constraint C. Initially, C is the falsified clause (satisfied term), called the *conflict clause (term)*. The solver finds the existential (universal) literal in C that was assigned last by QBCP, and the antecedent clause (term) R responsible for this assignment. A new constraint is derived by resolving C and R and applying universal (existential) reduction. This is done repeatedly until the resulting constraint C is *asserting*. A clause (term) S is asserting if there is a unique existential (universal) literal $\ell \in S$ with maximum decision level among literals in S, its decision level is greater than 0, the corresponding decision variable is existential (universal), and every universal (existential) variable $y \in var(S)$ such that $y \prec var(\ell)$ is assigned at a lower decision level (an asserting constraint becomes unit after backtracking). Once an asserting constraint has been found, it is added to the solver's

2. We do not consider the pure literal rule as part of QBCP.

set of constraints. Finally, QCDCL *backtracks*, undoing variable assignments until reaching a decision level computed from the learned constraint.

Example 1. We present a run of QCDCL on a simple PCNF formula Φ shown below.

$$\Phi = \forall x \exists y \exists z.(x \vee \neg y) \wedge (y \vee \neg z) \wedge (\neg x \vee z)$$

We use the notation $x \stackrel{d}{=} c$ to denote that variable x that is assigned value c by decision. If x is assigned c by unit propagation, we write $x = c$. A possible run of QCDCL on Φ looks as follows. The formula does not contain any unit clauses, so QCDCL has to make a decision, and the variable x is the only variable eligible for a decision. Setting $x \stackrel{d}{=} c$ satisfies the first clause and turns the third clause into the unit clause (z). Setting $z = 1$ turns the second clause into the unit clause (y), and setting $y = 1$ satisfies the matrix. This allows us to derive $(x \wedge z \wedge y)$ as an initial term using the *model generation rule* (Giunchiglia et al., 2006). By applying existential reduction we can derive the unit term (x) as a learned term. QCDCL then backtracks to decision level 0 and assigns $x = 0$. This turns the first clause into the unit clause $(\neg y)$ and results in the assignment $y = 0$. Now the second clause simplifies to $(\neg z)$ and the algorithm assigns $z = 0$. The resulting assignment satisfies the matrix and we derive $(\neg x \wedge \neg y \wedge \neg z)$ as an initial term. Existential reduction leads to the unit term $(\neg x)$ which is resolved with the term (x) responsible for assigning $x = 0$. The resolvent is the empty term and QCDCL returns *true*. □

4. QCDCL with Dependency Learning

We now describe how to modify QCDCL to support dependency learning. First, the solver

Algorithm 1 QCDCL with Dependency Learning

```
1: procedure SOLVE( )
2:     D = ∅
3:     while TRUE do
4:         conflict = QBCP()
5:         if conflict == NONE then
6:             DECIDE()
7:         else
8:             constraint, btlevel = ANALYZECONFLICT(conflict)
9:             if constraint != NONE then
10:                if ISEMPTY(constraint) then
11:                    return ISTERM(constraint)
12:                else
13:                    ADDLEARNEDCONSTRAINT(constraint)
14:                end if
15:            end if
16:            BACKTRACK(btlevel)
17:        end if
18:    end while
19: end procedure
```

Algorithm 2 Conflict Analysis with Dependency Learning

```
1: procedure analyzeConflict(conflict)
2:     constraint = getConflictConstraint(conflict)
3:     while not asserting(constraint) do
4:         pivot = getPivot(constraint)
5:         reason = getAntecedent(pivot)
6:         if existsResolvent(constraint, reason, pivot) then
7:             constraint = resolve(constraint, reason, pivot)
8:             constraint = reduce(constraint)
9:         else
10:            illegal_merges = illegalMerges(constraint, reason, pivot)
11:            D = D ∪ { (v, pivot) : v ∈ illegal_merges }
12:            return none, decisionLevel(pivot)
13:        end if
14:    end while
15:    btlevel = getBacktrackLevel(constraint)
16:    return constraint, btlevel
17: end procedure
```

maintains a set $D \subseteq \{ (x, y) : x \prec y \}$ of variable dependencies. Second, both QBCP and the decision rule must be modified as follows:

- qbcp() uses universal and existential reduction relative to D. Universal reduction relative to D removes each universal variable u from a clause C such that there is no existential variable $e \in var(C)$ with $(u, e) \in D$ (existential reduction relative to D is defined dually).
- decide() may assign any variable y such that there is no unassigned variable x with $(x, y) \in D$ (note that $(x, y) \in D$ implies $x \prec y$).

These rules correspond to how DepQBF uses the dependency relation D computed by a dependency scheme in propagation and decisions (Biere & Lonsing, 2010). Unlike DepQBF, QCDCL with dependency learning does *not* use the generalized reduction rules during conflict analysis (resolve and reduce in lines 7 and 8 refer to resolution and reduction as defined in Figure 1). As a consequence, the algorithm cannot always construct a learned constraint during conflict analysis (see Example 2 below). Such cases are dealt with in lines 9 through 12 of analyzeConflict (Algorithm 2):

- existsResolvent(*constraint*, *reason*, *pivot*) determines if the resolvent of *constraint* and *reason* exists.
- If this is not the case, there has to be a variable v (universal for clauses, existential for terms) satisfying the following condition: $v \prec$ *pivot* and there exists a literal $\ell \in$ *constraint* with $var(\ell) = v$ and $\overline{\ell} \in$ *reason*. The set of such variables is computed by illegalMerges. For each such variable, a new dependency is added to D. No learned constraint is returned by conflict analysis, and the backtrack level (*btlevel*) is set so as to cancel the decision level at which *pivot* was assigned.

The criteria for a constraint to be asserting must also be slightly adapted: a clause (term) S is asserting with respect to D if there is a unique existential (universal) literal $\ell \in S$ with maximum decision level among literals in S, its decision level is greater than 0, the corresponding decision variable is existential (universal), and every universal (existential) variable $y \in var(S)$ such that $(y, var(\ell)) \in D$ is assigned (again, this corresponds to the definition of asserting constraints used in DepQBF (Lonsing, 2012, p.119)). Finally, in the main QCDCL loop, we have to implement a case distinction to account for the fact that conflict analysis may not return a constraint (line 9).

Example 2. We revisit the PCNF formula Φ from Example 1 to illustrate a run of QCDCL with dependency learning.

$$\Phi = \forall x \exists y \exists z.(x \vee \neg y) \wedge (y \vee \neg z) \wedge (\neg x \vee z)$$

Initially, the set D of learned dependencies is empty. Accordingly, universal reduction relative to D would simplify the first clause to $(\neg y)$ and the third clause to (z). The algorithm thus assigns $y = 0$ and $z = 1$, falsifying the second clause. Conflict analysis first resolves the second clause with the third clause (which is responsible for propagating the last literal in the falsified clause) to obtain the clause $(\neg x \vee y)$. Since universal reduction is performed according to the prefix order during conflict analysis, the literal $\neg x$ cannot be removed from this clause even though there is no dependency of y on x. Next, conflict analysis attempts to resolve $(\neg x \vee y)$ with $(x \vee \neg y)$. These clauses do not have a resolvent in long-distance Q-resolution since $x \prec y$. Variable x is identified as involved in an illegal merge, the dependency (x, y) is added to D, and the solver backtracks before decision level 0 (the level where $\neg y$ was propagated), undoing all assignments. Because of the learned dependency $(x, y) \in D$ the first clause can no longer be simplified by universal reduction, but the third clause still simplifies to (z) and $z = 1$ is propagated. This simplifies the second clause to (y) and $y = 1$ is propagated. Thus the first clause becomes (x) and universal reduction results in a conflict. Conflict analysis resolves the first and second clause to derive $(x \vee \neg z)$. Because $x \prec z$ universal reduction cannot simplify this clause. Conflict analysis then attempts to resolve $(x \vee \neg z)$ and $(\neg x \vee z)$. These clauses do not have a resolvent and x is identified as the cause of an illegal merge. The dependency (x, z) is added to D and the solver backtracks to remove decision level 0 where z was propagated. Since $D = \{(x, y), (x, z)\}$ now contains all possible dependencies of Φ, QCDCL with dependency learning behaves exactly like ordinary QCDCL from that point onward.

4.1 Soundness and Termination

Soundness of QCDCL with dependency learning is an immediate consequence of the following observation.

Observation 1. *Every constraint learned by QCDCL with dependency learning can be derived from the input formula by long-distance Q-resolution or Q-consensus.*

To prove termination, we argue that the algorithm learns a new constraint or a new dependency after each conflict. Just as in QCDCL, every learned constraint is asserting, so learning does not introduce duplicate constraints.

Observation 2. *QCDCL with dependency learning never learns a constraint already present in the database.*

The only additional argument required to prove termination is one that tells us that the algorithm cannot indefinitely "learn" the same dependencies.

Lemma 1. *If QCDCL with dependency learning does not learn a constraint during conflict analysis, it learns a new dependency.*

Proof. To simplify the presentation, we are only going to consider clause learning (the proof for term learning is analogous). We first establish an invariant of intermediate clauses derived during conflict analysis: they are empty under the partial assignment obtained by backtracking to the last literal in the assignment that falsifies an existential literal in the clause. Formally, let C be a clause and let $\sigma = (\ell_1, \ldots, \ell_k)$ be an assignment. We define $\mathit{last}_C(\sigma) = \max(\{\, i : 1 \leq i \leq k \text{ and } \overline{\ell_i} \in C, \mathit{var}(\ell_i) \in E \,\} \cup \{0\})$, where E is the set of existential variables of the input PCNF formula, and let $\sigma_C = (\ell_1, \ldots, \ell_{\mathit{last}_C(\sigma)})$. In particular, if $\mathit{last}_C(\sigma) = 0$ then σ_C is empty.

We now prove the following claim: if σ is an assignment that falsifies a clause, then, for every intermediate clause C constructed during conflict analysis, $C[\sigma_C]$ is empty after universal reduction. The proof is by induction on the number of resolution steps in the derivation of C. If C is the conflict clause then $C[\sigma]$ reduces to the empty clause. That means $C[\sigma_C]$ can only contain universal literals and can also be reduced to the empty clause by universal reduction. Suppose C is the result of resolving clauses C' and R and applying universal reduction, where C' is an intermediate clause derived during conflict analysis and R is a clause that triggered unit propagation. The induction hypothesis tells us that $C'[\sigma_{C'}]$ reduces to the empty clause. Since the pivot literal ℓ is chosen to be the last existential literal falsified in C', we must have $\sigma_{C'} = (\ell_1, \ldots, \ell_k)$ where $\ell_k = \overline{\ell}$. Let $\tau = (\ell_1, \ldots, \ell_{k-1})$. We must have $C'[\tau] = U' \cup \{\ell\}$, where U' is a purely universal clause. Because R is responsible for propagating $\overline{\ell}$, we further must have $R[\tau] = U'' \cup \{\overline{\ell}\}$, where U'' again is a purely universal clause. It follows that their resolvent $C[\tau] = (C' \setminus \{\ell\})[\tau] \cup (R \setminus \{\overline{\ell}\})[\tau] = U' \cup U''$ is a purely universal clause. Since τ is a prefix of σ, it follows that $C[\sigma_C]$ is a purely universal clause as well and therefore empty after universal reduction. This proves the claim.

We proceed to prove the lemma. If the algorithm does not learn a clause during conflict analysis, this must be due to a failed attempt at resolving an intermediate clause C with a clause R responsible for unit propagation. That is, if e is the existential pivot variable, there must be a universal variable $u \prec e$ such that $u \in \mathit{var}(C) \cap \mathit{var}(R)$ and $\{u, \neg u\} \subseteq C \cup R$. Towards a contradiction, suppose that $(u, e) \in D$. Let σ denote the assignment that caused the conflict and assume without loss of generality that $\{u, e\} \subseteq R$ and $\{\neg u, \neg e\} \subseteq C$. Since R caused propagation of e but $(u, e) \in D$, the variable u must have been assigned before e and $\neg u \in \sigma$. As the pivot $\neg e$ is the last existential literal falsified in C, it follows that $\neg u \in \sigma_C$. Because $\neg u \in C$, this implies that the assignment σ_C satisfies C, in contradiction with the claim proved above. □

For a formula with n variables the number of dependencies is $O(n^2)$ and the number of distinct constraints is 2^{2n+1}. QCDCL runs into a conflict at least every n variable assignments, so Observation 2 and Lemma 1 imply termination.

Theorem 1. *QCDCL with dependency learning is sound and terminating.*

5. Experiments

To see whether dependency learning works in practice, we implemented a QCDCL solver that supports this technique named Qute.[3] We evaluated the performance of Qute in several experiments. First, we measured the number of instances solved by Qute on the union of benchmark sets from the 2016–2018 QBF Evaluations (Pulina, 2016). We compare these numbers with those of the best performing publicly available solvers for each input type. In a second experiment, we computed the dependency sets given by the standard dependency scheme (Samer & Szeider, 2009; Biere & Lonsing, 2009) and the reflexive resolution-path dependency scheme (Van Gelder, 2011; Slivovsky & Szeider, 2016) for preprocessed instances, and compared their sizes to the number of dependencies learned by Qute. Finally, we revisit an instance family which is known to be hard to solve for QCDCL (Janota, 2016) and show they pose no challenge to Qute. In fact, we reinforce the last experimental result by a formal proof that QCDCL with dependency learning can indeed solve instances from this family efficiently. For our experiments, we used a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux.

5.1 Decision Heuristic, Restart Strategy, and Constraint Deletion Policy

We briefly describe a few design choices for key components of Qute. The exact values of parameters (parameter names are shown in italics) used in the experiments are listed in Appendix A.

- We rely on a version of the variable move-to-front (VMTF) heuristic for selecting decision variables (Ryan, 2004; Biere & Fröhlich, 2015). VMTF maintains a list of variables and selects a decision variable that is closest to the head of the list. Upon learning a new constraint, variables occurring in the constraint are moved to the front of the list.

- Restarts are determined by a simple inner-outer restart scheme (Biere, 2008). A restart is triggered every time the conflict counter reaches a number referred to as the *inner restart distance*. After every restart, the inner restart distance is multiplied by a fixed *restart multiplier* and the conflict counter is reset. After a number of restarts corresponding to the *outer restart distance*, the inner restart distance is reset and the outer restart distance is multiplied by the *restart multiplier*.

- Qute keeps limits on the number of learned clauses and terms, respectively. Upon hitting the limit for clauses or terms, the corresponding constraints are ordered lexicographically according to their literal blocks distance (LBD) (Audemard & Simon, 2009) in increasing order and activity (Eén & Sörensson, 2003) in decreasing order. A fraction (determined by the *clause deletion ratio* and *term deletion ratio*) of these constraints is then deleted starting from the back of the list, skipping constraints that are locked because they are the antecedent of a literal on the current trail. The limit on the number of learned clauses or terms is then increased by a fixed constant (the *clause database increment* and *term database increment*, respectively).

3. `http://github.com/perebor/qute`

Table 1: Instances from the 2016–2018 QBF Evaluation prenex non-CNF (QCIR) benchmark sets solved within 10 minutes.

solver	total	sat	unsat
QuAbS	705	313	392
CQESTO	695	310	385
Qute	681	315	366
GhostQ	681	296	385
Qfun	663	296	367
Qute (no DL)	617	281	336
RAReQS	518	218	300

5.2 Solved Instances for QBF Evaluation 2016–2018 Benchmark Sets

In our first experiment, we used the prenex non-CNF (QCIR, Jordan, Klieber, & Seidl, 2016) benchmark sets from the 2016–2018 QBF Evaluation, consisting in total of 1240 formulas. Time and memory limits were set to 10 minutes and 4 GB, respectively. The results are summarized in Table 1 and Figure 2. Qute's performance is competitive with other state-of-the-art circuit solvers, and this appears to be in large part due to dependency learning: when dependency learning is deactivated, the number of solved instances drops significantly.

It is folklore within the QBF community that the number of quantifier alternations has a strong influence on solver performance. Generally speaking, expansion/abstraction solvers tend to do better on instances with few alternations, whereas QCDCL solvers are at an advantage on instances with many alternations. Standard benchmark sets contain many instances with only a single alternation (Lonsing & Egly, 2018), presumably because many problems of interest can be encoded in such formulas. Figure 3 shows the number of solved prenex non-CNF instances broken down by the number of quantifier alternations. While the number of instances with up to 100 alternations solved by Qute with dependency learning is slightly subpar (even compared to Qute without dependency learning), dependency learning shines when it comes to the subset of instances with the highest number (100+) of quantifier alternations.[4] This is also clearly visible in Figure 4, which shows the runtimes of Qute with and without dependency learning for individual instances. On average, dependency learning incurs a slight performance penalty for instances solved by both configurations but leads to many more solved instances among those with at least 100 quantifier alternations.

Many of these formulas have a number of quantifier alternations that is close to the overall number of variables. Apparently, most of the corresponding variable dependencies are spurious, and dependency learning allows Qute to ignore them.

4. This matches recent experimental results on portfolio-based algorithm selection for QCIR, where selectors favored Qute with dependency learning over other solvers for instances with many quantifier alternations (Hoos, Peitl, Slivovsky, & Szeider, 2018).

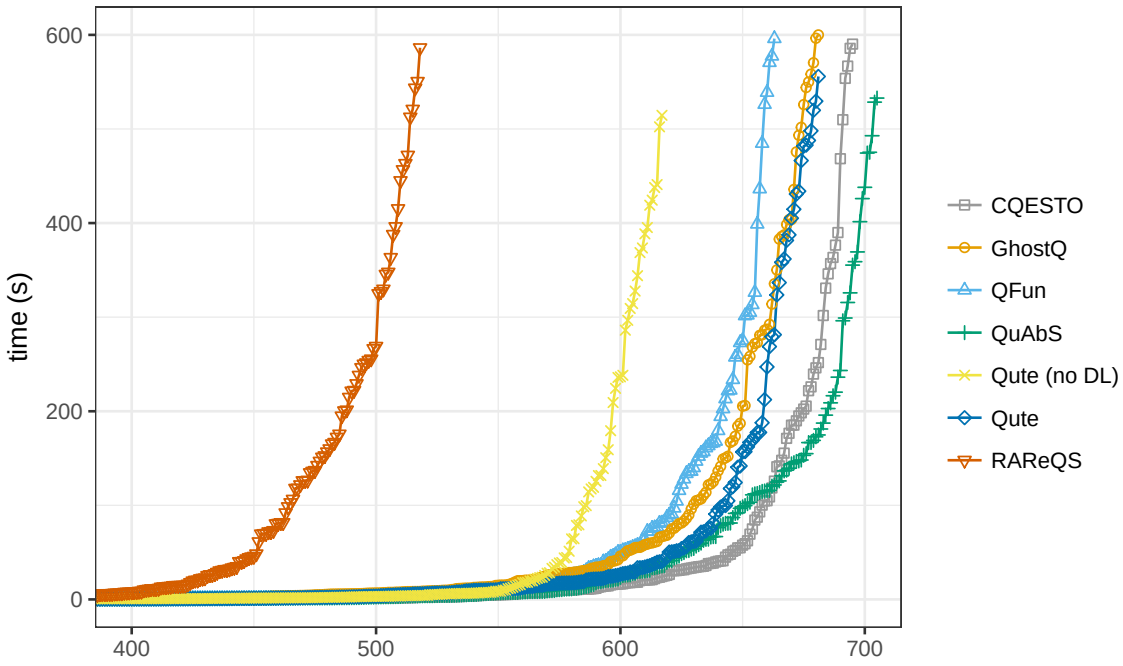

Figure 2: Solved instances from the 2016–2018 QBF Evaluation prenex non-CNF (QCIR) benchmark sets (x-axis) sorted by runtime (y-axis).

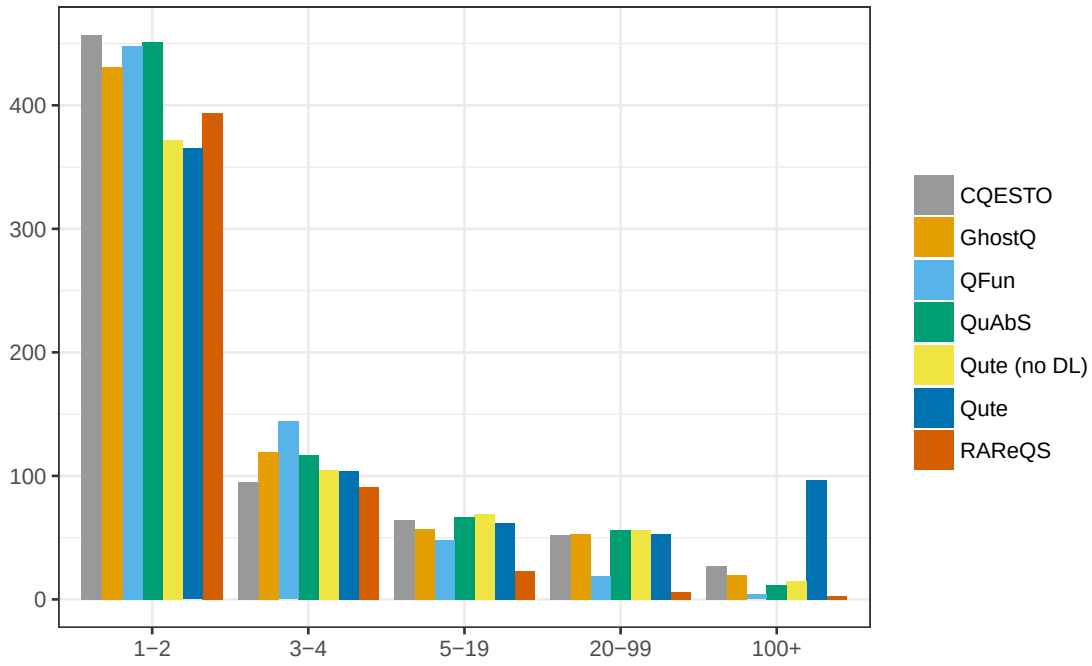

Figure 3: Solved instances from the 2016–2018 QBF Evaluation Prenex non-CNF (QCIR) benchmark sets (y-axis) by number of quantifier alternations (x-axis).

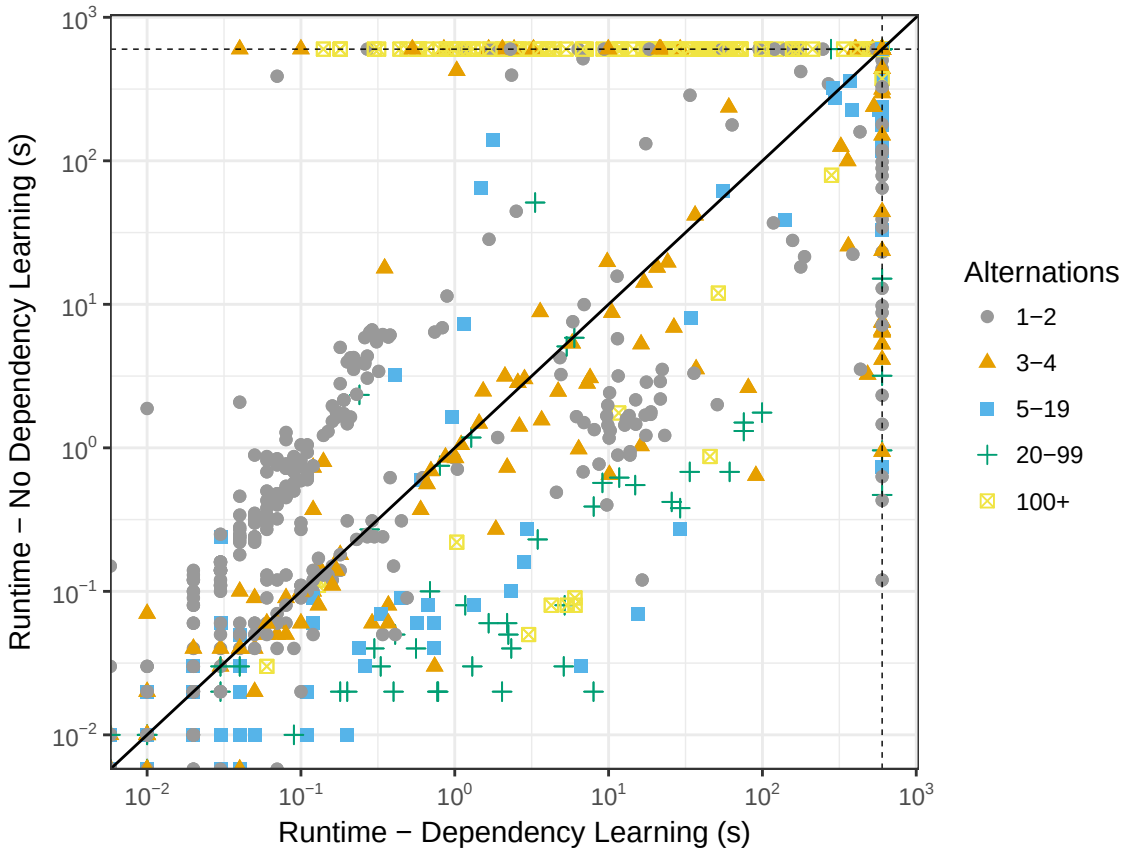

Figure 4: Runtimes of Qute with and without dependency learning on the 2016–2018 QBF Evaluation Prenex non-CNF (QCIR) benchmark sets, by number of quantifier alternations.

Table 2: Instances from the QBF Evaluation 2016–2018 prenex CNF (PCNF) benchmark sets solved within 10 minutes without preprocessing (left) and with preprocessing using HQSPre (right). Configurations labeled with +CR use partial circuit reconstruction.

solver	total	sat	unsat
GhostQ	752	350	402
Qute+CR	712	315	397
Qute (no DL)+CR	667	313	354
DepQBF	637	259	378
CAQE	549	216	333
RAReQS	510	152	358
Qute	499	166	333
Qute (no DL)	488	172	316

solver	total	sat	unsat
CAQE	978	442	536
RAReQS	945	410	535
DepQBF	888	396	492
Qute+CR	871	374	497
Qute (no DL)+CR	866	377	489
Qute (no DL)	858	381	477
GhostQ	833	369	464
Qute	827	347	480

For our second experiment, we used the prenex CNF (PCNF) benchmark sets from the 2016–2018 QBF Evaluations consisting of 1314 instances. Time and memory limits were again set to 10 minutes and 4 GB. We performed this experiment twice: with and without preprocessing using HQSPre (Wimmer et al., 2017). In order not to introduce variance in overall runtime through preprocessing, each instance was preprocessed only once, and solvers were run on the preprocessed instances with a timeout corresponding to the overall timeout minus the time spent on preprocessing.

Since Qute shows good performance on QCIR instances, we included configurations that perform partial circuit reconstruction using *qcir-conv*[5] and then solve the resulting QCIR instance.

The results obtained without using HQSPre are shown on the left-hand side of Table 2. When using qcir-conv, Qute solves significantly more instances with dependency learning than without dependency learning. Without qcir-conv, the difference is less pronounced, but dependency learning remains beneficial. Overall, we see that circuit reconstruction (also used internally by GhostQ (Klieber et al., 2010) substantially increases the performance of Qute.

The results including preprocessing with HQSPre are shown on the right-hand side of Table 2. The power of the preprocessor strikes the eye: any solver/configuration solves more instances when paired with HQSPre, than any other without it. While dependency learning does not seem to provide as much as in the previous cases, it is still part of the best-performing configuration of Qute.

5.3 Learned Dependencies Compared to Dependency Relations

To get an idea of how well QCDCL with dependency learning can exploit independence, we compared the number of dependencies learned by Qute with the number of standard and

5. `http://www.cs.cmu.edu/~wklieber/qcir-conv.py`

Table 3: Learned dependencies, standard dependencies, and reflexive resolution-path dependencies for instances preprocessed by HQSPre, as a fraction of trivial dependencies.

dependencies	mean	median	variance
standard	0.929	1.000	0.029
resolution-path	0.628	0.798	0.139
learned	0.033	0.007	0.004

resolution-path dependencies for instances from the PCNF benchmark set after preprocessing with HQSPre. We only considered instances with at least one quantifier alternation after preprocessing. Qute was run with a 10 minute timeout (excluding preprocessing). If an instance was not solved, we used the number of dependencies learned within that time limit.[6]

Summary statistics are shown in Table 3. On average, the standard dependency scheme only provides a mild improvement over trivial dependencies. The reflexive resolution-path dependency scheme does better, but the set of trivial dependencies it can identify as spurious is still small in many cases. The fraction of learned dependencies is much smaller than either dependency relation on average, and the median fraction of trivial dependencies learned is even below 1%.

This indicates that proof search in QCDCL with dependency learning is less constrained than in QCDCL with either dependency scheme: since QCDCL is allowed to branch on a variable x only if every variable that x depends on has already been assigned, decision heuristics are likely to have a larger pool of variables to choose from if fewer dependencies are present.

5.4 Dependency Learning on Hard Instances for QCDCL

For our third experiment, we chose a family of instances $\mathtt{CR}_n$ recently used to show that ordinary QCDCL does not simulate tree-like Q-resolution (Janota, 2016). Since the hardness of these formulas is tied to QCDCL not propagating across quantifier levels, they represent natural test cases for QCDCL with dependency learning. We recorded the number of backtracks required to solve $\mathtt{CR}_n$ by Qute with and without dependency learning, for $n \in \{1, ..., 50\}$. As a reference, we used DepQBF.[7] For this experiment, we kept the memory limit of 4 GB but increased the timeout to one hour. The results are summarized in Figure 5. As one would expect, Qute without dependency learning and DepQBF were only able to solve instances up to $n = 7$ and $n = 8$, respectively. Furthermore, it is evident

6. We cannot rule out that, for unsolved instances, Qute would have to learn a larger fraction of trivial dependencies before terminating. However, the solver tends to learn most dependencies at the beginning of a run, with the fraction of learned trivial dependencies quickly converging to a value that does not increase much until termination.

7. For the sake of comparing with Qute in prefix mode, we disabled features recently added to DepQBF such as dynamic quantified blocked clause elimination (Lonsing et al., 2015) and oracle calls to the expansion-based solver Nenofex.

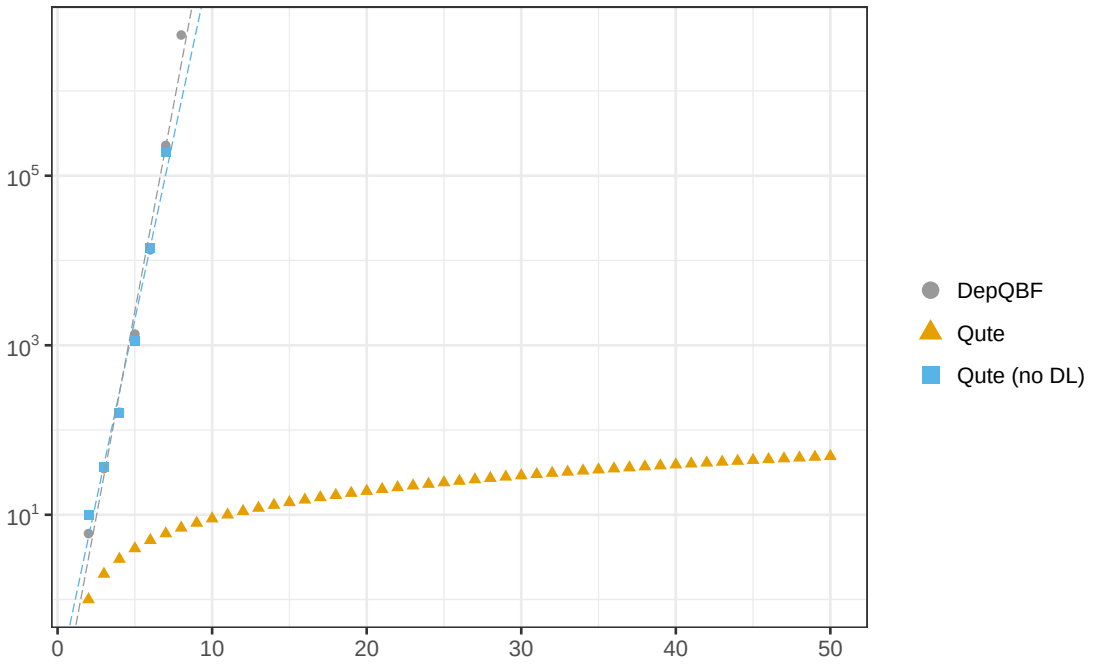

Figure 5: Backtracks for instances $\mathtt{CR}_n$ based on the Completion Principle (Janota, 2016), as a function of n.

from the plot that the number of backtracks grows exponentially with n for both solvers. By contrast, Qute with dependency learning was able to solve all instances within the timeout. In the next section, we will formally prove that QCDCL with dependency learning can find short proofs of the formulas $\mathtt{CR}_n$ for all n.

6. An Exponential Speedup over QCDCL

We will now show that there is a run of QCDCL with dependency learning on $\mathtt{CR}_n$ that terminates in time polynomial in n. The corresponding short proof of $\mathtt{CR}_n$ that we identify is, up to a symmetry, the one presented by Janota (2016). Let us first recall the definition of $\mathtt{CR}_n$.

Definition 1. *Let $n \in \mathbb{N}$, let $\mathcal{X} = \{ x_{ij} : 1 \leq i, j \leq n \}$, and $\mathcal{L} = \{ a_i, b_i : 1 \leq i \leq n \}$ be sets of variables. The formula $\mathtt{CR}_n$ has the prefix $\exists\mathcal{X}\forall z\exists\mathcal{L}$, and the matrix consisting of the following clauses:*

$$A_{ij} = x_{ij} \vee z \vee a_i \qquad \text{for } 1 \leq i, j \leq n$$
$$B_{ij} = \neg x_{ij} \vee \neg z \vee b_j \qquad \text{for } 1 \leq i, j \leq n$$
$$A = \bigvee_{i=1}^{n} \neg a_i \qquad B = \bigvee_{j=1}^{n} \neg b_j$$

Lemma 2. *Assume that the current trail of a QCDCL solver with dependency learning contains only literals from the set $\{a_1, \ldots, a_n\}$. Then, assigning any a_i that is not on the trail to false causes unit propagation to derive a conflict, and the clause $(z \vee a_i)$ is derived during conflict analysis.*

Proof. Assigning a_i to false causes the clause A_{ij} to propagate x_{ij} for $j = 1, \ldots, n$. In turn, assigning x_{ij} to true causes the clause B_{ij} to propagate b_j for $j = 1, \ldots, n$ (because there is no dependency of b_j on z). This causes a conflict with the clause B.

During conflict analysis, we will resolve B with the clauses B_{ij} on $b_1, \ldots, b_n$, effectively removing all $\neg b_j$ from B, and introducing $\neg x_{i1}, \ldots, \neg x_{in}$ and $\neg z$. At that point, $\neg z$ is trailing and can be reduced. Afterwards, we will resolve away all the $\neg x_{ij}$ using A_{ij}, and we will end up with the clause $(z \vee a_i)$ as required. Note that each intermediate clause contains at least two literals from the set $\{a_i, \neg b_1, \ldots, \neg b_n, \neg x_{i1}, \ldots, \neg x_{in}\}$ which are existential and assigned at the same decision level as a_i (which is the highest decision level) so none of these clauses can be asserting. □

Consider the following run of QCDCL with dependency learning. Each clause contains at least two existential literals, so (even with an empty dependency relation) no variable is assigned by unit propagation at decision level 0. As the first decision, the variable a_1 is assigned to false. By Lemma 2 the clause $(z \vee a_1)$ is derived during conflict analysis. This clause is asserting, so it is learned, the algorithm backtracks to decision level 0, and unit propagation sets a_1 to true. This is repeated for $a_1, \ldots, a_{n-1}$. During propagation of the assignment $a_1 \wedge \cdots \wedge a_{n-1}$ the clause A propagates $\neg a_n$, which by Lemma 2 again leads to a conflict and the derivation of the clause $(z \vee a_n)$ during conflict analysis. This time, because $\neg a_n$ is assigned at decision level 0, the clause $(z \vee a_n)$ is not asserting. Thus conflict analysis proceeds to resolve $(z \vee a_n)$ with A, and then subsequently with the clauses $(z \vee a_i)$ for $i = 1, \ldots, n-1$. This yields the unit clause (z), which is reduced to the empty clause, resulting in QCDCL terminating and outputting FALSE.

This run requires n invocations of propagation and conflict analysis. Since both unit propagation and conflict analysis can be carried out in polynomial time, we get a polynomial bound on the overall runtime.

Theorem 2. *QCDCL with dependency learning can solve $\mathtt{CR}_n$ in polynomial time.*

Since ordinary QCDCL requires time exponential in n to solve $\mathtt{CR}_n$ (Janota, 2016), dependency learning achieves an exponential speedup on these instances.

7. An Interpretation of Learned Dependencies

Dependencies in QBF naturally arise in a semantic context. The formula $\forall x \exists y . \varphi(x, y)$ says that we can choose a value for y, *depending on* x, such that $\varphi(x, y)$ evaluates to true. In other words, the assertion that this formula is true is equivalent to saying that there is a function f_y, which depends on x as its input, and which chooses values for y so that φ evaluates to true. Such a function is called a *model* of the formula, and one way of thinking about dependencies in a given formula is to think about the dependencies exploited by a model/all models of the formula. For instance, in this example, it could be the case that a constant function f_y that does not exploit the information about x at all suffices to make

$\varphi(x, y)$ evaluate to true—in such a case we would say that the dependency of y on x, while present *syntactically*, is spurious *semantically*.

QCDCL with dependency learning does not directly extract semantic information about the formula it is solving in the sense outlined in the previous paragraph—it simply relaxes the rules of QCDCL, and refines the relaxation (by learning a dependency) to avoid an illegal merge during constraint learning. Nevertheless, there is something that can be said about the relationship between learned and "actual", semantic dependencies of a formula. In this section, we formalize what we mean by semantic dependencies, and study how they relate to learned dependencies.

Our notion of semantic dependencies is based on changes (or lack thereof) in the truth value of a formula when certain variables are shifted around in the prefix. As an example, consider the formula

$$\Phi = \forall u \forall x \exists z \forall v \exists y \exists e.\varphi$$

for some matrix φ. We define the y-x shift of Φ as the formula

$$\Phi' = \forall u \exists y \forall x \exists z \forall v \exists e.\varphi,$$

i.e., the variable y is moved "just in front of" x, maintaining the relative order of other variables. This is formalized in the definition below.

Definition 2. *Let $\Phi = Q_1 x_1 \dots Q_n x_n.\varphi$ be a QBF, and let $x = x_k$ and $y = x_m$, $k < m$. Let $g_k^m(j)$ for $j = 1, \dots, n$ be defined in the following way:*

$$g_k^m(j) = \begin{cases} j & \text{if } j < k \text{ or } m < j \\ m & \text{if } j = k \\ j - 1 & \text{otherwise} \end{cases}$$

The y-x shift of Φ *is the formula $\Phi_x^y = Q_{g_k^m(1)} x_{g_k^m(1)} \dots Q_{g_k^m(n)} x_{g_k^m(n)}.\varphi$.*

Notice that the mapping g_k^m is a permutation of the set $\{1, \dots, n\}$, and the prefix $Q_{g_k^m(1)} x_{g_k^m(1)} \dots Q_{g_k^m(n)} x_{g_k^m(n)}$ is what results from $Q_1 x_1 \dots Q_n x_n$ when y is shifted just in front of x. This notion can naturally be extended to shifting in front of arbitrary sets of variables.

Definition 3. *Let $\Phi = Q_1 x_1 \dots Q_n x_n.\varphi$ be a QBF, let $y = x_m$, and $\emptyset \neq X \subseteq var(\Phi)$. Let $k = \min_{x_j \in X} j < m$. The y-X* shift of Φ *is the formula $\Phi_X^y = \Phi_{x_k}^y$.*

For a given variable y, any set of variables on which a model for y is allowed to depend (syntactically), i.e., a set of variables left of y and of opposite quantifier type, will be called a *syntactic dependency set*.

Definition 4. *Let $\Phi = \mathcal{Q}.\varphi$ be a QBF, $X \subseteq var(\Phi)$, and $y \in var(\Phi) \setminus X$. If $x \prec_\Phi y$ for each $x \in X$, then we say that X is a* syntactic dependency set *of y. If $X = \{x\}$, we say that x is a* syntactic dependency *of y.*

Finally, we define *potential* and *critical* semantic dependencies. Intuitively, X is a *critical dependency set* of y if moving y in front of X in the prefix makes the formula change its truth value. X is a *potential dependency set* of y if the player who owns y loses when y is shifted in front of X, but does not necessarily win when y is in its original place. This is akin to a necessary condition—X must be left of y if there should be a chance to win, but maybe not even that is sufficient. Every critical dependency set is also a potential dependency set.

Definition 5. *Let $\Phi = \mathcal{Q}.\varphi$ be a QBF, y one of its existential (universal) variables, and X a syntactic dependency set of y. If Φ^y_X is false (true), then we say that X is a* potential dependency set *of y. If, moreover, Φ is true (false), then we say that X is a* critical dependency set *of y. If $X = \{ x \}$, we say that x is a* potential *or* critical dependency *of y, respectively.*

We note that in the above definition, a critical (potential) dependency is a critical (potential) dependency set of cardinality 1, and that this is different from being an element of a larger critical (potential) dependency set. It can be easily seen that any superset of a critical (potential) dependency set is also a critical (potential) dependency set, but the same does not necessarily hold for subsets.

Example 3. Consider the formula

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

Clearly Φ is true, as is witnessed by the model $f_y(x) = x$, and x is a syntactic dependency of y. If we shift y in front of x, we get the formula $\Phi^y_x = \exists y \forall x\ (x \vee \neg y) \wedge (\neg x \vee y)$, which is false, witnessed by the countermodel $f_x(y) = \neg y$. Hence, we conclude that x is a critical dependency of y.

Example 4. Consider the formula

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

If we shift y in front of x, we get the formula $\Phi^y_x = \exists y \forall x\ (x \vee \neg y) \wedge (x \vee y)$, which is false, witnessed by the countermodel $f_x(y) = 0$. Hence, x is a potential dependency of y. However, in this case Φ itself is false, and so x is not a critical dependency of y.

The results of this section show that learning certain seemingly necessary dependencies can be avoided, and provide a characterization of learned dependencies and the context in which they are learned. Theorem 3 says that QCDCL with dependency learning can solve formulas with many critical dependencies (critical dependency sets of size 1) without learning any of them. Theorem 4 on the other hand says that any learned dependency must be contained in a potential dependency set in a restriction of the input formula.

Theorem 3. *For every $n \in \mathbb{N}$ there is a QBF Φ_n with $O(n^2)$ variables and $\Omega(n^2)$ critical dependencies, and a run of QCDCL with dependency learning on Φ_n that terminates in polynomial time and learns no dependency.*

Proof. Let $\Phi_n = \mathtt{CR}_n$ (see Definition 1). The required run of QCDCL with dependency learning has already been presented in the previous section. It remains to show that there are indeed the required critical dependencies.

Let $1 \leq i_0, j_0 \leq n$, and consider the QBF $(\Phi_n)^z_{x_{ij}}$, which results from swapping x_{ij} and z in the prefix of Φ_n. It can be verified that the following set of (mostly constant) functions is a model of $(\Phi_n)^z_{x_{ij}}$:

$$\begin{aligned} x_{i_0 j} &= 1 && \text{for } j \neq j_0, \\ x_{i j_0} &= 0 && \text{for } i \neq i_0, \\ x_{i_0 j_0} &= \neg z, \\ a_i = b_j &= 1 && \text{for } i \neq i_0 \wedge j \neq j_0, \\ a_{i_0} = b_{j_0} &= 0, \end{aligned}$$

and hence $(\Phi_n)^z_{x_{ij}}$ is true. Since Φ_n is false, we get that $x_{i_0 j_0}$ is a critical dependency of z, and there are n^2 choices of i_0, j_0. □

Theorem 4. *Assume QCDCL with dependency learning is solving the formula Φ and learns a dependency of a variable y on a variable x. Let τ be the current trail assignment, and let σ be τ restricted to literals of the same quantifier type as y. Then there is $X \subseteq var(\Phi)$, such that $x \in X$ and X is a potential dependency set of y in $\Phi[\sigma]$.*

Proof. Without loss of generality, let us assume that x is universal and y is existential. Consider QCDCL with dependency learning in the state when it learns a new dependency of y on x. The learned dependency stems from a failed attempt at long-distance resolution of two clauses R, the clause that became unit during search, and C, the intermediate learned clause, derived from Φ, over the pivot y, without loss of generality $y \in R$. Let τ be the trail assignment at the point when the resolution of R and C is attempted, i.e., up to but excluding y, and let σ be τ restricted to existential literals. Since R was unit under τ and propagated y, and C is the intermediate learned clause, which must contain only universal literals after the application of $\tau \cup \{y\}$ (cf. proof of Lemma 1), we have that y is the only existential literal in $R[\sigma]$ and $\overline{y}$ is the only existential literal in $C[\sigma]$. The reason why long-distance resolution of R and C fails is that there is at least one universal variable $x' \prec y$ blocking the resolution, i.e., without loss of generality, $x' \in R$ and $\overline{x'} \in C$. Let X be the set of all universal variables blocking the resolution of R and C, clearly a syntactic dependency set of y. We will show that σ and X satisfy the stated conditions.

First, by definition of dependency learning, the learned dependency x is in the set X. We need to prove that X is a potential dependency set of y in $\Phi[\sigma]$, i.e., we need to show that $\Phi[\sigma]^y_X$ is false. We will do that by restricting the long-distance Q-resolution derivations of the clauses R and C, which are implicitly generated by QCDCL, by the assignment σ. Since neither R, nor C are satisfied by σ, and σ only assigns existential variables, the restricted derivations derive clauses $R' \subseteq R[\sigma]$ and $C' \subseteq C[\sigma]$ (Peitl et al., 2016, Lemma 1). If no literal on y is present in either of R' and C', we already have a long-distance Q-resolution derivation of a purely universal clause, and so $\Phi[\sigma]$, and therefore also $\Phi[\sigma]^y_X$, is false (shifting an existential variable to the left can only make a true formula false, not the other way around). Otherwise, consider the shifted prefix in $\Phi[\sigma]^y_X$. With this prefix, we

can reduce all literals that are possibly blocking the resolution of R' and C', and hence we can derive the empty clause. Since shifting the existential variable y left in the prefix does not invalidate any previous reductions or resolution steps, we have a valid long-distance Q-resolution refutation of $\Phi[\sigma]^y_X$, and hence X is really a potential dependency set of y in $\Phi[\sigma]$. □

Example 5. Theorem 4 can be illustrated with the following example. Consider the formula

$$\Phi = \exists z \forall x \exists y.\ (z \vee x \vee \neg y) \wedge (z \vee \neg x \vee y),$$

which essentially says *if not z, then $y = x$*. One of the models of Φ is $f_z = 1$, which remains a model even if we shift y in front of x. Hence, x is not even a potential dependency of y. However, assume QCDCL with dependency learning is solving this formula and starts with the decision $z \overset{d}{=} 0$. Subsequently, because no dependencies have been learned yet, unit propagation w.l.o.g. sets $y = 1$, and the clause $(z \vee x \vee \overline{y})$ becomes falsified by universal reduction. Conflict analysis will attempt to resolve the (only) two clauses of Φ, which is however prevented by x, and a dependency of y on x will have to be learned.

Theorem 4 says there is a σ and a potential dependency set X of y in $\Phi[\sigma]$. In particular, $\sigma = \neg z$, the decision made by the solver, and $X = \{ x \}$. Therefore, we could also paraphrase Theorem 4 in the following way: if a dependency of y on x is learned, then even if perhaps x is not contained in any potential dependency set of y in the original formula Φ, it is definitely contained in a potential dependency set of Φ restricted by the "current trail assignment", which can be thought of as the actual formula that the solver is solving at the moment of learning the dependency. In this example, at the moment of learning the dependency of y on x, z is assigned to false, and hence the restricted formula says $y = x$, and so learning the dependency is justified.

Note that this example also shows that it is unlikely that we could provide a stronger condition for learned dependencies than Theorem 4, because even when solving a fairly trivial formula with no potential dependency sets dependencies can still be learned.

8. Discussion

Dependency learning has several advantages over the use of dependency schemes within QCDCL. First, it is arguably easier to implement. The integration of the standard dependency scheme with DepQBF required the development of data structures for the succinct representation of standard dependencies (Biere & Lonsing, 2009), and no such compact representation is currently known for the resolution-path dependency scheme. By contrast, dependency sets in Qute are encoded in arrays containing a list of variables. Second, our experiments indicate that proof search is less constrained since Qute typically learns only a fraction of the dependencies computed by dependency schemes. Third, by keeping long-distance Q-resolution as the underlying proof system, QCDCL with dependency learning is amenable to a simple correctness proof and enjoys linear-time strategy extraction.

Blinkhorn and Beyersdorff (2017) offered a strong proof-theoretic argument in favor QCDCL with dependency schemes over "vanilla" QCDCL by proving an exponential separation of Q($\mathrm{D}^{\mathrm{rrs}}$)-resolution and ordinary Q-resolution, where Q($\mathrm{D}^{\mathrm{rrs}}$)-resolution is the proof system used by QCDCL with the reflexive resolution-path dependency scheme $\mathrm{D}^{\mathrm{rrs}}$

and a form of constraint learning that avoids long-distance resolution (e.g. Lonsing, Egly, & Van Gelder, 2013). However, this separation was proved against a class of formulas introduced by Kleine Büning, Karpinski, & Flögel (1995) which is known to have short proofs in long-distance Q-resolution (Egly et al., 2013), so it does not speak about the relative strength of QCDCL with the resolution-path dependency scheme and QCDCL with dependency learning.

In our experiments, Qute performed much better when presented with non-CNF input. In particular, dependency learning was most effective on the prenex non-CNF (QCIR) benchmark set, accounting for a 10% increase in the number of solved instances. Even for PCNF formulas, the best configuration(s) used tools for partially recovering circuit structure from CNF. This is consistent with the fact that dependency learning had a more limited effect on Qute when preprocessing was used, since preprocessing is known to adversely affect circuit reconstruction (Goultiaeva & Bacchus, 2013), and so it would impair Qute's best-performing configurations. Whether this bias towards non-CNF representations is inherent to QCDCL with dependency learning or an artifact of other design choices implemented in our solver remains to be seen.

Sometimes, the additional freedom afforded by dependency learning appears to be detrimental to performance. In particular, this seems to be the case when quantifier alternations reflect strict temporal or logical dependencies, such as in formulas encoding two-player games. Freed from the restrictions normally imposed by the quantifier prefix, decision heuristics originally devised for SAT steer Qute into regions of the search space that make little sense semantically and force it to learn many useless constraints and dependencies. We see two ways of addressing this issue. First, we want to design new decision heuristics that take the quantifier prefix into account, possibly by penalizing out-of-order assignments. Second, we plan to develop techniques for initializing learned dependencies with a small set of dependencies that might help steer proof search in the right direction. For instance, Qute uses Tseitin conversion to obtain a set of initial clauses and terms from non-CNF (QCIR) instances. We found that assigning a Tseitin variable before a variable used in its definition often results in learning a dependency, so that it pays off to simply include dependencies of a Tseitin variable on the variables used in its definition from the start. For similar reasons, it might make sense to include dependencies induced by implicit variable definitions (Lang & Marquis, 2008). Efficient techniques for detecting implicit variable definitions have been developed for preprocessing in propositional model counting (Lagniez et al., 2016; Ivrii et al., 2016).

In recent research (Peitl, Slivovsky, & Szeider, 2019), we showed that dependency learning can also be fruitfully combined with dependency schemes: when the solver is about to learn a missing dependency, we dynamically query a dependency scheme to discover independent variables, which are then excluded from the set of learned dependencies (cf. the *else* branch of Algorithm 2 starting on line 10).

Finally, it is worth noting that dependency learning supports the removal of learned dependencies just as well as their addition. Although we did not encounter instances where the set of learned dependencies grows so large that it significantly affects performance, it is possible that the management of learned dependencies causes an overhead during longer solver runs. For that reason, removing learned dependencies at regular intervals in analogy to constraint deletion might be beneficial.

Acknowledgments

We want to thank Florian Lonsing for helpful discussions related to QCDCL and the reviewers for their thoughtful comments on our manuscript. This research was kindly supported by FWF grants P27721 and W1255-N23.

Appendix A. Solver Parameters for Experiments in Section 5

parameter	value	description
`--decision-heuristic`	VMTF	Decision heuristic, see Section 5.
`--restarts`	inner-outer	Restart strategy, see Section 5.
`--inner-restart-distance`	250	Initial number of conflicts before restart.
`--outer-restart-distance`	20	Initial number of restarts before outer restart.
`--restart-multiplier`	2.5	Multiplier for increasing inner and outer restart distance.
`--initial-clause-DB-size`	1000	Initial limit on learned clauses.
`--initial-term-DB-size`	4000	Initial limit on learned terms.
`--clause-DB-increment`	1000	Upon reaching the current limit on the number of learned clauses, increase the limit by this value.
`--term-DB-increment`	500	Upon reaching the current limit on the number of learned terms, increase the limit by this value.
`--clause-removal-ratio`	0.4	Fraction of learned clauses to delete upon reaching the current limit.
`--term-removal-ratio`	0.3	Fraction of learned terms to delete upon reaching the current limit.
`--LBD-threshold`	5	Only delete constraints whose LBD is greater than this value.
`--constraint-activity-decay`	0.99	Multiply constraint activities with this value after each conflict.
`--constraint-activity-inc`	-2	Add this value to the activity score of a constraint whenever it is seen during conflict analysis.
`--dependency-learning`	all	Add all variables involved in an illegal merge as learned dependencies.
`--phase-heuristic`	invJW	Heuristic for choosing the assignment of a decision variable.
`--model-generation`	depQBF	Use DepQBF-style model generation for PCNF instances: pick the first satisfying literal in each clause, with a preference for existential literals.

References

Audemard, G., & Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. In Boutilier, C. (Ed.), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 399–404.

Balabanov, V., & Jiang, J. R. (2012). Unified QBF certification and its applications. *Formal Methods in System Design*, *41*(1), 45–65.

Balabanov, V., Jiang, J. R., Janota, M., & Widl, M. (2015). Efficient extraction of QBF (counter)models from long-distance resolution proofs. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 3694–3701. AAAI Press.

Biere, A. (2004). Resolve and expand. In *Proceedings of SAT 2004 (Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada)*, pp. 59–70.

Biere, A. (2008). Adaptive restart strategies for conflict driven SAT solvers. In Kleine Büning, H., & Zhao, X. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, Vol. 4996 of *Lecture Notes in Computer Science*, pp. 28–33. Springer Verlag.

Biere, A., & Fröhlich, A. (2015). Evaluating CDCL variable scoring schemes. In Heule, M., & Weaver, S. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, Vol. 9340 of *Lecture Notes in Computer Science*, pp. 405–422. Springer Verlag.

Biere, A., & Lonsing, F. (2010). Integrating dependency schemes in search-based QBF solvers. In Strichman, O., & Szeider, S. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2010*, Vol. 6175 of *Lecture Notes in Computer Science*, pp. 158–171. Springer Verlag.

Biere, A., & Lonsing, F. (2009). A compact representation for syntactic dependencies in QBFs. In Kullmann, O. (Ed.), *Theory and Applications of Satisfiability Testing - SAT 2009*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 398–411. Springer Verlag.

Blinkhorn, J., & Beyersdorff, O. (2017). Shortening QBF proofs with dependency schemes. In Gaspers, S., & Walsh, T. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2017*, Vol. 10491 of *Lecture Notes in Computer Science*, pp. 263–280. Springer Verlag.

Cadoli, M., Schaerf, M., Giovanardi, A., & Giovanardi, M. (2002). An algorithm to evaluate Quantified Boolean Formulae and its experimental evaluation. *Journal of Automated Reasoning*, *28*(2).

Eén, N., & Sörensson, N. (2003). An extensible SAT-solver. In Giunchiglia, E., & Tacchella, A. (Eds.), *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, Vol. 2919 of *Lecture Notes in Computer Science*, pp. 502–518. Springer Verlag.

Egly, U., Lonsing, F., & Widl, M. (2013). Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In McMillan, K. L., Middeldorp, A., & Voronkov, A. (Eds.), *Logic for Programming, Artificial Intelligence, and Reasoning - LPAR 2013*, Vol. 8312 of *Lecture Notes in Computer Science*, pp. 291–308. Springer Verlag.

Faymonville, P., Finkbeiner, B., Rabe, M. N., & Tentrup, L. (2017). Encodings of bounded synthesis. In Legay, A., & Margaria, T. (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017*, Vol. 10205 of *Lecture Notes in Computer Science*, pp. 354–370.

Giunchiglia, E., Narizzano, M., & Tacchella, A. (2006). Clause/term resolution and learning in the evaluation of Quantified Boolean Formulas. *J. Artif. Intell. Res.*, *26*, 371–416.

Goultiaeva, A., & Bacchus, F. (2013). Recovering and utilizing partial duality in QBF. In Järvisalo, M., & Van Gelder, A. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2013*, Vol. 7962 of *Lecture Notes in Computer Science*, pp. 83–99. Springer Verlag.

Heule, M. J. H., Kullmann, O., & Marek, V. W. (2016). Solving and verifying the Boolean Pythagorean Triples problem via cube-and-conquer. In Creignou, N., & Berre, D. L. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, Vol. 9710 of *Lecture Notes in Computer Science*, pp. 228–245. Springer Verlag.

Hoos, H. H., Peitl, T., Slivovsky, F., & Szeider, S. (2018). Portfolio-based algorithm selection for circuit QBFs. In Hooker, J. N. (Ed.), *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018*, Vol. 11008 of *Lecture Notes in Computer Science*, pp. 195–209. Springer Verlag.

Ivrii, A., Malik, S., Meel, K. S., & Vardi, M. Y. (2016). On computing minimal independent support and its applications to sampling and counting. *Constraints*, *21*(1), 41–58.

Janota, M. (2016). On Q-resolution and CDCL QBF solving. In Creignou, N., & Berre, D. L. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, Vol. 9710 of *Lecture Notes in Computer Science*, pp. 402–418. Springer Verlag.

Janota, M., Klieber, W., Marques-Silva, J., & Clarke, E. M. (2012). Solving QBF with counterexample guided refinement. In Cimatti, A., & Sebastiani, R. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2012*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 114–128. Springer Verlag.

Janota, M., & Marques-Silva, J. (2015). Solving QBF by clause selection. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pp. 325–331. AAAI Press.

Jordan, C., Klieber, W., & Seidl, M. (2016). Non-CNF QBF solving with QCIR. In Darwiche, A. (Ed.), *Beyond NP, Papers from the 2016 AAAI Workshop.*, Vol. WS-16-05 of *AAAI Workshops*. AAAI Press.

Kleine Büning, H., Karpinski, M., & Flögel, A. (1995). Resolution for quantified Boolean formulas. *Information and Computation*, *117*(1), 12–18.

Klieber, W., Sapra, S., Gao, S., & Clarke, E. M. (2010). A non-prenex, non-clausal QBF solver with game-state learning. In Strichman, O., & Szeider, S. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2010*, Vol. 6175 of *Lecture Notes in Computer Science*, pp. 128–142. Springer Verlag.

Lagniez, J., Lonca, E., & Marquis, P. (2016). Improving model counting by leveraging definability. In Kambhampati, S. (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 751–757. IJCAI/AAAI Press.

Lang, J., & Marquis, P. (2008). On propositional definability. *Artificial Intelligence*, *172*(8-9), 991–1017.

Lonsing, F. (2012). *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. Ph.D. thesis, Johannes Kepler University, Linz, Austria.

Lonsing, F., Bacchus, F., Biere, A., Egly, U., & Seidl, M. (2015). Enhancing search-based QBF solving by dynamic blocked clause elimination. In Davis, M., Fehnker, A., McIver, A., & Voronkov, A. (Eds.), *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, Vol. 9450 of *Lecture Notes in Computer Science*, pp. 418–433. Springer Verlag.

Lonsing, F., & Biere, A. (2008). Nenofex: Expanding NNF for QBF solving. In Büning, H. K., & Zhao, X. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2008*, Vol. 4996 of *Lecture Notes in Computer Science*, pp. 196–210. Springer Verlag.

Lonsing, F., & Egly, U. (2018). Evaluating QBF solvers: Quantifier alternations matter. In Hooker, J. N. (Ed.), *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018*, Vol. 11008 of *Lecture Notes in Computer Science*, pp. 276–294. Springer Verlag.

Lonsing, F., Egly, U., & Van Gelder, A. (2013). Efficient clause learning for quantified Boolean formulas via QBF pseudo unit propagation. In Järvisalo, M., & Van Gelder, A. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2013*, Vol. 7962 of *Lecture Notes in Computer Science*, pp. 100–115. Springer Verlag.

Malik, S., & Zhang, L. (2009). Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, *52*(8), 76–82.

Marques-Silva, J. P., Lynce, I., & Malik, S. (2009). Conflict-driven clause learning SAT solvers. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, pp. 131–153. IOS Press.

Meel, K. S., Vardi, M. Y., Chakraborty, S., Fremont, D. J., Seshia, S. A., Fried, D., Ivrii, A., & Malik, S. (2016). Constrained sampling and counting: Universal hashing meets SAT solving. In Darwiche, A. (Ed.), *Beyond NP, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 12, 2016.*, Vol. WS-16-05 of *AAAI Workshops*. AAAI Press.

Peitl, T., Slivovsky, F., & Szeider, S. (2016). Long distance Q-resolution with dependency schemes. In Creignou, N., & Berre, D. L. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July*

5-8, 2016, Proceedings, Vol. 9710 of *Lecture Notes in Computer Science*, pp. 500–518. Springer Verlag.

Peitl, T., Slivovsky, F., & Szeider, S. (2019). Combining resolution-path dependencies with dependency learning. In Janota, M., & Lynce, I. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Lecture Notes in Computer Science. Springer Verlag. To appear.

Pulina, L. (2016). The ninth QBF solvers evaluation - preliminary report. In Lonsing, F., & Seidl, M. (Eds.), *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016).*, Vol. 1719 of *CEUR Workshop Proceedings*, pp. 1–13. CEUR-WS.org.

Rabe, M. N., & Tentrup, L. (2015). CAQE: A certifying QBF solver. In Kaivola, R., & Wahl, T. (Eds.), *Formal Methods in Computer-Aided Design - FMCAD 2015*, pp. 136–143. IEEE Computer Soc.

Ryan, L. (2004). Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University.

Samer, M., & Szeider, S. (2009). Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, *42*(1), 77–97.

Scholl, C., & Pigorsch, F. (2016). The QBF solver AIGSolve. In Lonsing, F., & Seidl, M. (Eds.), *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016)*, Vol. 1719 of *CEUR Workshop Proceedings*, pp. 55–62. CEUR-WS.org.

Slivovsky, F., & Szeider, S. (2016). Soundness of Q-resolution with dependency schemes. *Theoretical Computer Science*, *612*, 83–101.

Tentrup, L. (2016). Non-prenex QBF solving using abstraction. In Creignou, N., & Berre, D. L. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2016*, Vol. 9710 of *Lecture Notes in Computer Science*, pp. 393–401. Springer Verlag.

Van Gelder, A. (2011). Variable independence and resolution paths for quantified Boolean formulas. In Lee, J. (Ed.), *Principles and Practice of Constraint Programming - CP 2011*, Vol. 6876 of *Lecture Notes in Computer Science*, pp. 789–803. Springer Verlag.

Vizel, Y., Weissenbacher, G., & Malik, S. (2015). Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, *103*(11), 2021–2035.

Wimmer, R., Reimer, S., Marin, P., & Becker, B. (2017). HQSpre - an effective preprocessor for QBF and DQBF. In Legay, A., & Margaria, T. (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017*, Vol. 10205 of *Lecture Notes in Computer Science*, pp. 373–390.

Zhang, L., & Malik, S. (2002). Conflict driven learning in a quantified Boolean satisfiability solver. In Pileggi, L. T., & Kuehlmann, A. (Eds.), *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pp. 442–449. ACM / IEEE Computer Society.