Combining Resolution-Path Dependencies with Dependency Learning

Tomáš Peitl(✉), Friedrich Slivovsky, and Stefan Szeider

Algorithms and Complexity Group, TU Wien, Vienna, Austria
{peitl,fs,sz}@ac.tuwien.ac.at

Abstract. We present the first practical implementation of the reflexive resolution-path dependency scheme in a QBF solver. Unlike in DepQBF, which uses the less general standard dependency scheme, we do not compute the dependency relation upfront, but instead query relevant dependencies on demand during dependency conflicts, when the solver is about to learn a missing dependency. Thus, our approach is fundamentally tied to dependency learning, and shows that the two techniques for dependency analysis can be fruitfully combined. As a byproduct, we propose a quasilinear-time algorithm to compute all resolution-path dependencies of a given variable. Experimental results on the QBF library confirm the viability of our technique and identify families of formulas where the speedup is particularly promising.

1 Introduction

Dependency analysis is a state-of-the-art technique in QBF solving, in which the QBF solver attempts to identify spurious syntactic dependencies between variables and by doing so simplify the quantifier prefix. The historically older approach to dependency analysis are *dependency schemes* [14], first implemented in the QCDCL (Quantified Conflict-Driven Clause/Cube/Constraint Learning) solver DepQBF [2]. A dependency scheme is a mapping that, given a formula, identifies pairs of variables that are syntactic dependencies according to the quantifier prefix, but can in fact be safely ignored. DepQBF employs the standard dependency scheme in order to identify pairs of variables that are guaranteed to be independent. A more recent idea, implemented in the QCDCL solver Qute [11], is *dependency learning*, where the solver speculatively assumes all pairs of variables to be independent, and updates the information whenever it proves wrong during search.

Since the dawn of DepQBF and its use of the standard dependency scheme, one of the main open questions in QBF dependency analysis has been whether stronger dependency schemes can be utilized as well. Since DepQBF uses tailor-made data structures to efficiently compute the standard dependency scheme [3], one cannot answer this question by simply substituting a different dependency

This research was partially supported by FWF grants P27721 and W1255-N23.

M. Janota and I. Lynce (Eds.): SAT 2019, LNCS 11628, pp. 306–318, 2019.
https://doi.org/10.1007/978-3-030-24258-9_22

scheme into DepQBF. Of particular interest would be an efficient implementation of the *reflexive resolution-path dependency scheme* [16,17], the strongest known tractable sound one.

The main issue with implementing dependency schemes is that the number of pairs of variables in the dependency relation can in the worst case be quadratic, which turns out to be impractical for a large number of relevant formulas. We therefore take a different approach, and only compute parts of the dependency relation on demand. We do this during *dependency conflicts*, a state of the solver unique to QCDCL with dependency learning, in which the solver attempts to perform a resolution step, but fails due to universal literals (in the case of clause resolution) left of the pivot variable appearing in different polarities in the two clauses and thus blocking the resolution step. When the solver encounters a dependency conflict, it would normally have to learn a new dependency. Instead, we compute the dependencies of the pivot variable and filter out any blocking variables that are actually independent. If it turns out that no blocking variables remain, the resolution step can be carried out, otherwise a dependency on one or more of the remaining blocking variables is learned.

While it is known that resolution paths are equivalent to directed paths in the implication graph of the formula [15], using this result directly would require us to perform one search for each blocking variable, resulting in an overall quadratic running time. Instead, we show that all dependencies of a given variable can be found by searching for *widest paths* in a weighted variant of the implication graph. This is a well-studied problem that can be solved efficiently, for instance in overall quasilinear time using a variant of Dijkstra's algorithm [9].

We implemented the dependency scheme in the QBF solver Qute, and evaluated our implementation on the entire QBF Library [5], preprocessed by the preprocessor HQSpre [18]. We observed a modest increase in the total number of solved instances. We also identified families of formulas on which the use of the dependency scheme appears to be particularly beneficial.

2 Preliminaries

A *CNF formula* is a finite conjunction $C_1 \wedge \cdots \wedge C_m$ of clauses, a *clause* is a finite disjunction $(\ell_1 \vee \cdots \vee \ell_k)$ of literals, and a *literal* is a variable x or a negated variable $\overline{x}$. We will also refer to *terms* (also known as *cubes*), which are finite conjunctions of literals. Whenever convenient, we consider clauses and terms as sets of literals, and CNF formulas as sets of clauses. The length of a CNF formula $\varphi = C_1 \wedge \cdots \wedge C_m$ is defined as $\|\varphi\| = \sum_{i=1}^{m} |C_m|$.

We consider QBFs in Prenex Conjunctive Normal Form (PCNF), i.e., formulas $\mathcal{F} = \mathcal{Q}.\varphi$ consisting of a (quantifier) prefix $\mathcal{Q}$ and a propositional CNF formula φ, called the *matrix* of $\mathcal{F}$. The *prefix* is a sequence $\mathcal{Q} = Q_1 x_1 \ldots Q_n x_n$, where $Q_i \in \{\forall, \exists\}$ is a universal or existential quantifier and the x_i are (universal or existential) variables. The *depth* of a variable x_i is defined as $\delta(x_i) = i$, the depth of a literal ℓ is $\delta(\ell) = \delta(\mathrm{var}(\ell))$. If $\delta(x) < \delta(y)$ we say that x is left of y, y is right of x, and we write $R_{\mathcal{F}}(x) = \{\, v \in \mathrm{var}(\mathcal{F}) \mid \delta(x) < \delta(v) \,\}$, and

$x_i \prec_{\mathcal{F}} x_j$ if $1 \leq i < j \leq n$ and $Q_i \neq Q_j$, dropping the subscript if the formula $\mathcal{F}$ is understood. The length of a PCNF formula $\mathcal{F} = \mathcal{Q}.\varphi$ is defined as $\|\mathcal{F}\| = \|\varphi\|$.

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 $\mathrm{var}(x) = \mathrm{var}(\overline{x}) = x$ to denote the variable associated with a literal and let $\mathrm{var}(C) = \{\, \mathrm{var}(\ell) \mid \ell \in C \,\}$ if C is a clause (term), $\mathrm{var}(\varphi) = \bigcup_{C \in \varphi} \mathrm{var}(C)$ if φ is a CNF formula, and $\mathrm{var}(\mathcal{F}) = \mathrm{var}(\varphi)$ if $\mathcal{F} = \mathcal{Q}.\varphi$ is a PCNF formula.

The semantics of a PCNF formula Φ are 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.

2.1 QCDCL and Q-Resolution

We briefly review QCDCL and Q-resolution [10], 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 [19], the formal definition shown in Fig. 1 was given only recently [1]. 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 analogue of universal reduction for terms is called *existential reduction*).

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

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 $e \prec \mathrm{var}(\ell_u)$ 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 $\neg e \vee C_2$. Each variable u for which $\ell_u \in C_1$ and $\overline{\ell_u} \in C_2$ is said to be *merged over* e in this resolution step.

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

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.

Fig. 1. Long-distance Q-resolution.

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 [1]. 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.

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 [4,6]. Accordingly, the solver outputs TRUE if the empty term is learned, and FALSE if the empty clause is learned.

3 Resolution-Path Dependency Scheme

The reflexive resolution path dependency scheme detects spurious dependencies of PCNF formulas based on *resolution paths* [16,17].

Definition 1 (Resolution Path). *Let $\mathcal{F} = \mathcal{Q}.\varphi$ be a PCNF formula and let X be a set of variables. A* resolution path *from ℓ_1 to ℓ_{2k} in $\mathcal{F}$ is a sequence $\pi = \ell_1, \ldots, \ell_{2k}$ of literals satisfying the following properties:*

1. *for all $i \in \{1, \ldots, k\}$, there is a $C_i \in \varphi$ such that $\ell_{2i-1}, \ell_{2i} \in C_i$,*
2. *for all $i \in \{1, \ldots, k\}$, $var(\ell_{2i-1}) \neq var(\ell_{2i})$,*
3. *for all $i \in \{1, \ldots, k-1\}$, $\overline{\ell_{2i}} = \ell_{2i+1}$.*

If additionally

4. *for all $i \in \{1, \ldots, k-1\}$, $\{\ell_{2i}, \ell_{2i+1}\} \subseteq X \cup \overline{X}$,*

then we say that π is a resolution path via *X. If $\pi = \ell_1, \ldots, \ell_{2k}$ is a resolution path in $\mathcal{F}$ (via X), we say that ℓ_1 and ℓ_{2k} are* connected in *$\mathcal{F}$ (with respect to X). For every $i \in \{1, \ldots, k-1\}$ we say that π* goes through *$\mathrm{var}(\ell_{2i})$ and $var(\ell_{2i}), 1 \leq i < k$ are the* connecting variables *of π.*

Definition 2 (Proper Resolution Path). *Let ℓ, ℓ' be two literals of a PCNF formula $\mathcal{F}$ such that $\delta(\ell') < \delta(\ell)$. A resolution path from ℓ to ℓ' is called* proper*, if it is a resolution path via $R_{\mathcal{F}}(var(\ell')) \cap var_{\exists}(\mathcal{F})$. If there is a proper resolution path from ℓ to ℓ', we say that ℓ and ℓ' are* properly connected *(in $\mathcal{F}$).*

Resolution paths can be understood in terms of walks in the *implication graph* of a formula [15].

Definition 3 (Implication graph). *Let $\mathcal{F} = \mathcal{Q}.\varphi$ be a PCNF formula. The* implication graph *of $\mathcal{F}$, denoted by $IG(\mathcal{F})$ is the directed graph with vertex set $var(\mathcal{F}) \cup \overline{var(\mathcal{F})}$ and edge set $\{ (\overline{\ell}, \ell') \mid$ there is a $C \in \varphi$ such that $\ell, \ell' \in C$ and $\ell \neq \ell' \}$.*

Lemma 1 ([15]). *Let $\mathcal{F}$ be a PCNF formula and let $\ell, \ell' \in var(\mathcal{F}) \cup \overline{var(\mathcal{F})}$ be distinct literals. The following statements are equivalent:*

1. $\ell, \ell_1, \overline{\ell_1}, \ldots, \ell_k, \overline{\ell_k}, \ell'$ *is a resolution path from* ℓ *to* ℓ',
2. $\overline{\ell}, \ell_1, \ldots, \ell_k, \ell'$ *is a path in* $IG(\mathcal{F})$.

The resolution path dependency scheme identifies variables connected by a pair of resolution paths as potentially dependent on each other. We call a pair of variables connected in this way a *dependency pair*.

Definition 4 (Dependency pair). *Let* $\mathcal{F}$ *be a PCNF formula and* $x, y \in var(\mathcal{F})$. *We say* $\{x, y\}$ *is a* resolution-path dependency pair *of* $\mathcal{F}$ *with respect to* $X \subseteq var_\exists(\mathcal{F})$ *if at least one of the following conditions holds:*

- x *and* y, *as well as* $\neg x$ *and* $\neg y$, *are connected in* $\mathcal{F}$ *with respect to* X.
- x *and* $\neg y$, *as well as* $\neg x$ *and* y, *are connected in* $\mathcal{F}$ *with respect to* X.

It remains to determine the set X of variables with respect to which a pair x, y of variables needs to be connected to induce a dependency. For $x \prec_\mathcal{F} y$, the original *resolution-path dependency scheme* only included dependency pairs $\{x, y\}$ connected with respect to existential variables to the right of x, excluding x and y. It turns out that this dependency scheme can be used for reordering the quantifier prefix [15] but does not lead to a sound generalization of Q-resolution as required for use within a QCDCL-solver [16]. By dropping the restriction that x and y must not appear on the resolution paths inducing a dependency pair, we obtain the *reflexive resolution-path dependency scheme*, which yields a sound generalization of Q-resolution [16].

Definition 5 (Proper dependency pair). *Let* $\mathcal{F}$ *be a PCNF formula and* $x, y \in var(\mathcal{F})$, $\delta(x) < \delta(y)$. *We say* $\{x, y\}$ *is a* proper *resolution-path dependency pair of* $\mathcal{F}$ *if at least one of the following conditions holds:*

- x *and* y, *as well as* $\neg x$ *and* $\neg y$, *are properly connected in* $\mathcal{F}$.
- x *and* $\neg y$, *as well as* $\neg x$ *and* y, *are properly connected in* $\mathcal{F}$.

Definition 6. *The* reflexive resolution-path dependency scheme *is the mapping* $\mathcal{D}^{rrs}$ *that assigns to each PCNF formula* $\mathcal{F} = \mathcal{Q}.\varphi$ *the relation*

$$\mathcal{D}^{rrs}_\mathcal{F} = \{\, x \prec_\mathcal{F} y \mid \{x, y\} \text{ is a proper resolution-path dependency pair of } \mathcal{F} \,\}.$$

When $\mathcal{D}^{\mathrm{rrs}}$ is used in QCDCL solving, the solver learns clauses in a generalization of long-distance Q-resolution called LDQ($\mathcal{D}^{\mathrm{rrs}}$)-resolution. Figure 3 shows the proof rules of LDQ($\mathcal{D}^{\mathrm{rrs}}$)-resolution. Soundness of LDQ($\mathcal{D}^{\mathrm{rrs}}$)-resolution has been established by [12].

Theorem 1 (Corollary 3, [12]). *LDQ($\mathcal{D}^{rrs}$)-resolution is sound.*

We note that the soundness of the corresponding LDQ($\mathcal{D}^{\mathrm{rrs}}$)-consensus for terms still remains as an open problem. In our experiments with the proof system, we have been able to independently verify the truth value of all formulas by a different QBF solver.

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

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee e$ and $\neg e \vee C_1$, where e is an existential variable, the *(long-distance) resolution* rule can derive the clause $C_1 \vee C_2$, provided that $(u, e) \notin \mathcal{D}^{\text{rrs}}_{\mathcal{F}}$ for each universal variable u with $u \in C_1$ and $\overline{u} \in C_2$ (or vice versa), and that $C_1 \vee C_2$ does not contain an existential variable in both polarities.

$$\frac{C}{C \setminus \{u, \neg u\}}\ \text{(generalized } \forall\text{-reduction)}$$

The $\forall$*-reduction* rule derives the clause $C \setminus \{u, \neg u\}$ from C, where $u \in \text{var}(C)$ is a universal variable such that $(u, e) \notin \mathcal{D}^{\text{rrs}}_{\mathcal{F}}$ for every existential variable $e \in \text{var}(C)$.

Fig. 2. Derivation rules of LDQ($\mathcal{D}^{\text{rrs}}$)-resolution for a PCNF formula $\mathcal{F} = \mathcal{Q}.\varphi$.

4 Using Resolution-Path Dependencies in Practice

The major issue with implementing any dependency scheme for use in a QBF solver is the fact that the size of the dependency relation is inherently worst-case quadratic in the number of variables—all pairs of variables of opposite quantifier type potentially need to be stored. QBFs of interest often contain hundreds of thousands of variables, and therefore any procedure with quadratic complexity is infeasible. DepQBF overcomes this by identifying equivalence classes of variables with identical dependency information, and storing only one chunk of data per equivalence class [3]. This compressed form, however, is specifically tailored to the standard dependency scheme, and cannot directly be transferred to other dependency schemes.

4.1 Dynamically Applying $\mathcal{D}^{\text{rrs}}$

In order to avoid the quadratic blowup, we take a different approach. We do not aim at computing the entire dependency relation, but instead compute parts of it on demand, when a *dependency conflict* occurs.

Dependency conflicts in clause learning in QCDCL with dependency learning take place in the following way (in this entire section we focus on the case of clauses, but the case of term learning is dual): the solver attempts to resolve two clauses, C_1 and C_2, over a pivot variable e, but there is a non-empty set of universal variables U, such that

$$\forall u \in U\ u \prec e, (u \in C_1 \wedge \overline{u} \in C_2) \vee (\overline{u} \in C_1 \wedge u \in C_2).$$

These variables are blocking the resolution step, as is shown in the pseudocode snippet in Algorithm 1 (for a more thorough treatment of QCDCL with dependency learning we refer to [11]). The reason why this occurs is that the solver

mistakenly assumed e not to depend on any $u \in U$, and this erroneous assumption is now to be rectified by learning the dependency of e on at least one variable from U.

Algorithm 1. 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 // dependency conflict
10:            U = illegalMerges(constraint, reason, pivot)
11:            D = D ∪ { (v, pivot) | v ∈ U }
12:            return none, decisionLevel(pivot)
13:        end if
14:    end while
15:    btlevel = getBacktrackLevel(constraint)
16:    return constraint, btlevel
17: end procedure
```

We can conveniently insert a dynamically computed dependency scheme at this moment. Before any dependency of e is learned, the dependencies of e according to the dependency scheme are computed. Any $u \in U$ that turns out to be independent of e can be removed from the set of blocking variables. If everything in U is independent, no dependency needs to be learned, and conflict analysis can proceed by performing a resolution step in LDQ($\mathcal{D}^{\text{rrs}}$)-resolution, in which all $u \in U$ are merged over e. If some variables in U turn out to be actual dependencies of e, at least one of them has to be learned as usual. The modification to the conflict analysis process is shown in Algorithm 2.

The computed dependencies of e are then stored and re-used in any future dependency conflicts featuring e as the pivot variable, as well as in strengthening the reduction rule.

Soundness of QCDCL with dependency learning *and* the reflexive resolution-path dependency scheme follows from the soundness of long-distance Q($\mathcal{D}^{\text{rrs}}$)-resolution, the underlying proof system used by the algorithm.

4.2 Dynamically Computing $\mathcal{D}^{\text{rrs}}$

When computing resolution-path connections, it is natural to start with a variable v, and compute all variables which depend on v. This is because in this case, the set of connecting variables that can form proper resolution paths is

Algorithm 2. Conflict Analysis with DL and a Dependency Scheme

```
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 // dependency conflict
10:            U = illegalMerges(constraint, reason, pivot)
11:            rrs_deps[pivot] = getDependencies(pivot)
12:            U = U ∩ rrs_deps[pivot]
13:            if U = ∅ then
14:                goto 7
15:            else
16:                D = D ∪ { (v, pivot) | v ∈ U }
17:                return none, decisionLevel(pivot)
18:            end if
19:        end if
20:    end while
21:    btlevel = getBacktrackLevel(constraint)
22:    return constraint, btlevel
23: end procedure
```

fixed—all existential variables right of v are permitted—and the task of finding everything that depends on v is reducible to reachability in a single directed graph. However, since a dependency conflict may feature any number of blocking variables, we would potentially need to perform the search many times in order to check each dependency. It would be preferable to compute all dependencies of the pivot variable instead. However, since for every blocking variable $u \in U$, the set of allowed connecting variables may be different, we cannot reduce the task of finding all dependencies of the pivot e to just reachability in a single directed graph, and we need a different approach.[1]

Definition 7. *Let $\mathcal{F}$ be a PCNF formula, ℓ a literal of $\mathcal{F}$, and $w_\ell : var(\mathcal{F}) \cup \overline{var(\mathcal{F})} \to \mathbb{R} \cup \{\pm\infty\}$ the mapping defined by*

$$w_\ell(l) = \begin{cases} \infty & \text{if } l = \overline{\ell}, \\ \delta(\ell) & \text{if } l \neq \overline{\ell} \text{ and } var(l) \text{ is existential,} \\ -\infty & \text{otherwise.} \end{cases}$$

The depth-implication graph for $\mathcal{F}$ *at ℓ, denoted $DIG(\mathcal{F}, \ell)$ is the weighted version of $IG(\mathcal{F})$ where the weight of an edge (ℓ_1, ℓ_2) is defined as $w(\ell_1, \ell_2) = w_\ell(\ell_1)$.*

[1] This is the case regardless of the quantifier type of the pivot, the issue is that different targets in the set of blocking variables can be reached using different connecting variables.

For a path π in a weighted directed graph G, the *width* of π is defined as the minimum weight over all edges of π. The following theorem relates resolution paths in a formula with *widest paths* in its depth-implication graph.

Theorem 2. *Let ℓ, ℓ' be two literals of a PCNF formula $\mathcal{F}$ such that $\delta(\ell') < \delta(\ell)$. There is a proper resolution path from ℓ to ℓ' if, and only if, the widest path from $\overline{\ell}$ to ℓ' in DIG$(\mathcal{F}, \ell)$ has width larger than $\delta(\ell')$.*

Proof. Let $\pi = \ell, \ell_2, \ldots, \ell_{2k-1}, \ell'$ be a proper resolution path, and let $\pi' = \overline{\ell}, \ell_2, \ldots, \ell_{2k-2}, \ell'$ be the corresponding path in DIG$(\mathcal{F}, \ell)$ (by Lemma 1). The width of π' is defined as

$$\begin{aligned} w(\pi') &= \min\left\{w(\overline{\ell}, \ell_2), \ldots, w(\ell_{2k-2}, \ell')\right\} \\ &= \min\left\{w_\ell(\overline{\ell}), \ldots, w_\ell(\ell_{2k-2})\right\}. \end{aligned}$$

Since $w_\ell(\overline{\ell}) = \infty$ and π is proper and hence none of its connecting variables are universal, we have that $w(\pi') = \min\{\delta(\ell_2), \ldots, \delta(\ell_{2k-2})\} > \delta(\ell')$, where the inequality follows from π being proper.

Conversely, let $\pi' = \overline{\ell_1}, \ell_2, \ldots, \ell_k, \ell'$ be a path of width greater than $\delta(\ell')$, and let $\pi = \ell_1, \ell_2, \overline{\ell_2}, \ldots, \ell_k, \overline{\ell_k}, \ell'$ be the corresponding resolution path. Since $w(\pi') > \delta(\ell')$, no connecting variables in π can be universal, and they all have to be right of ℓ', hence π is proper. □

Naively applying the algorithm from [15] would result in an overall quadratic running time needed to determine all dependencies of a given variable v. Using Theorem 2 we can reduce the task to two searches for widest paths, and obtain a much more favourable time bound.

Theorem 3. *Given a variable v of a PCNF formula $\mathcal{F}$, all resolution-path dependencies, i.e., the set $\{ x \in var(\mathcal{F}) \mid (x, v) \in \mathcal{D}^{rrs}_{\mathcal{F}} \}$, can be computed in time $O\big(\|\mathcal{F}\| \log \|\mathcal{F}\|\big)$.*

Proof. In order to find out whether a given candidate variable x is a dependency of v, one has to determine whether there is a pair of proper resolution paths, either from v to x and from $\overline{v}$ to $\overline{x}$, or from v to $\overline{x}$ and from $\overline{v}$ to x. Theorem 2 tells us that the existence of proper resolution paths is equivalent to existence of wide paths. A generalization of Dijkstra's algorithm can compute widest paths from a single source to all destinations in a given graph in quasilinear time [9]. The key observation is that the entire computation is performed within two graphs, namely DIG$(\mathcal{F}, v)$ and DIG$(\mathcal{F}, \overline{v})$. By computing all widest paths from both v and $\overline{v}$, and then subsequently checking for which candidate variables x both polarities of x are reached by a wide enough path, we can find all dependencies of v.

By using the clause-splitting trick like in [15] we can, in linear time, obtain an equisatisfiable formula $\mathcal{F}'$ with var$(\mathcal{F}) \subseteq$ var$(\mathcal{F}')$ such that the resolution-path connections between variables of $\mathcal{F}$ are the same. Since $\mathcal{F}'$ has bounded clause size, we get that the number of edges in IG$(\mathcal{F}')$ is $O\big(\|\mathcal{F}'\|\big) = O\big(\|\mathcal{F}\|\big)$, and the stated running time is then simply the running time of Dijkstra's algorithm. □

5 Experiments

We modified the dependency-learning solver Qute so as to perform the procedure described above—when a dependency is about to be learned, resolution-path dependencies of the pivot variable are computed, and all blocking variables that turned out to be spurious dependencies are eliminated. Furthermore, the computed dependencies are kept for re-use in future dependency conflicts featuring the same pivot variable, as well as to be used in generalized ∀-reduction.

We evaluated our solver on a cluster of 16 machines each having two 10-core Intel Xeon E5-2640 v4, 2.40 GHz processors and 160 GB of RAM, running Ubuntu 16.04. We set the time limit to 900 s and the memory limit to 4 GB. As our benchmark set, we selected the QDIMACS instances available in the QBF Library[2] [5]. We first preprocessed them using the preprocessor HQSpre[3] [18] with a time limit of 400 seconds, resulting in a set of 14893 instances not solved by HQSpre. Out of these instances, we further identified the set of easy instances as those solved within 10 seconds by each of the following solvers: CaQE[4] 3.0.0 [13], DepQBF[5] 6.03 [2], QESTO[6] 1.0 [8], Qute[7] 1.1 [11], and RaReQS[8] 1.1 [7]. We decided to focus only on instances not solved by at least one of these solvers in under 10 s, as it arguably makes little sense to try and push state of the art for formulas that can already be solved in almost no time regardless of the choice of the solver. That left us with a set of 11262 instances.

Table 1 and Fig. 3 show the comparison between plain Qute and the version which implements the dependency scheme (Qute-$\mathcal{D}^{\text{rrs}}$). The version with the dependency scheme solved 176 (roughly 4.5%) more instances than the version without. The scatter plot in Fig. 3 deserves further attention. While the overall number of solved instances is higher for Qute-$\mathcal{D}^{\text{rrs}}$, the plot is skewed towards Qute-$\mathcal{D}^{\text{rrs}}$. We attribute this to a small overhead associated with the use of the dependency scheme, which is most apparent for the easiest formulas. The plot also shows that there are a few formulas solved by the plain version, but not by Qute-$\mathcal{D}^{\text{rrs}}$. This is only partly due to the additional time spent computing resolution paths, and is, in our opinion, in much larger part due to the heuristics being led off the right track towards a proof of the formula.

We found two families of instances where the increase in number of solved instances is even more significant, as is documented in Table 1. Particularly on the *matrix multiplication* and *reduction finding* benchmarks the dependency scheme provides a tremendous boost of performance, resulting in almost four times as many solved instances.

[2] http://www.qbflib.org/.
[3] https://projects.informatik.uni-freiburg.de/users/4.
[4] https://www.react.uni-saarland.de/tools/caqe.
[5] https://github.com/lonsing/depqbf.
[6] http://sat.inesc-id.pt/~mikolas/sw/qesto.
[7] https://github.com/perebor/qute.
[8] http://sat.inesc-id.pt/~mikolas/sw/areqs.

Table 1. Number of instances solved by plain Qute vs Qute using the reflexive resolution-path dependency scheme on the 'matrix multiplication' and 'reduction finding' families of formulas, as well as on all instances.

	MM-family	RF-family	all instances
# of instances	334	2269	11262
solved by Qute (SAT / UNSAT)	34 (4/30)	423 (140/283)	3959 (1467/2492)
solved by Qute-$\mathcal{D}^{\text{rrs}}$ (SAT / UNSAT)	123 (4/119)	484 (144/340)	4135 (1489/2646)

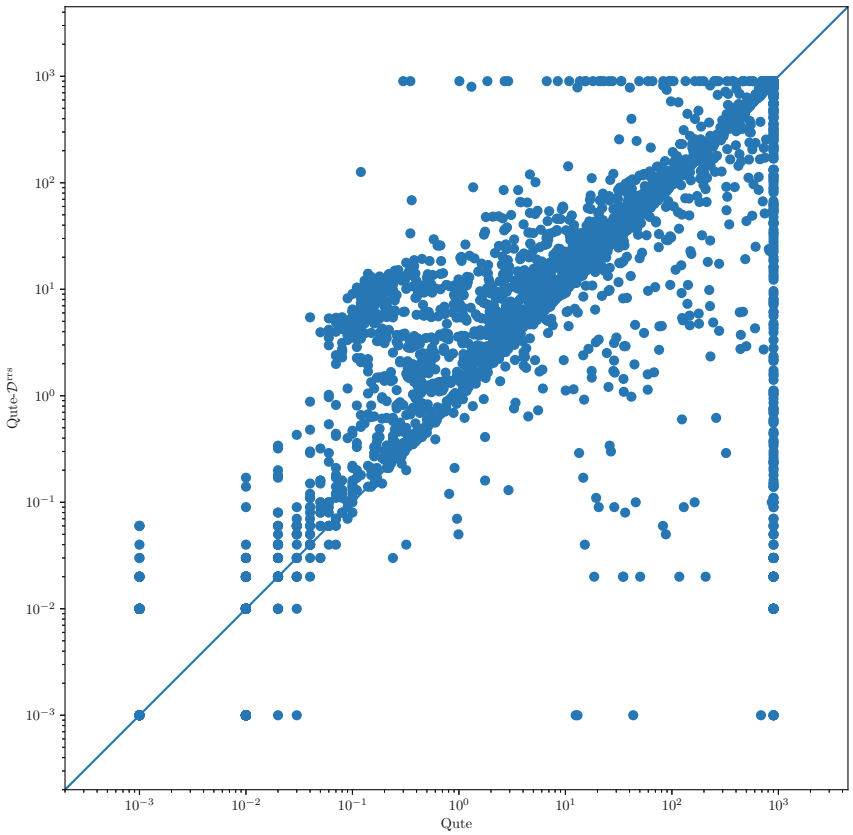

Fig. 3. Runtimes of Qute with and without $\mathcal{D}^{\text{rrs}}$on all instances.

6 Conclusion and Future Work

We presented the first practical implementation of $\mathcal{D}^{\text{rrs}}$ in a QBF solver. Thus, we have demonstrated that the strongest known tractable sound dependency scheme can be efficiently used in QBF solving. Our approach shows that dependency schemes can be fruitfully combined with dependency learning. Our algorithm for the computation of all resolution-path dependencies of a given variable may also be of independent interest.

While the additional prefix relaxation that comes from $\mathcal{D}^{\text{rrs}}$ is no cure-all for the hardness of QBF, we have found families of formulas where it provides a significant speedup. In particular, the use of the dependency scheme turned out very beneficial on the 'matrix multiplication' and 'reduction finding' classes, which are both practically relevant applications and further improvement using QBF would be valuable.

A possible direction for future work is to try to further improve the time bound of our algorithm for computing the resolution-path dependencies of a variable either by using data structures more suitable for this concrete scenario, or by preprocessing the formula. A succinct, possibly implicit, representation of $\mathcal{D}^{\text{rrs}}$ for use in other solver architectures would also be very interesting.

References

1. Balabanov, V., Jiang, J.H.R.: Unified QBF certification and its applications. Formal Methods Syst. Des. **41**(1), 45–65 (2012)
2. Lonsing, F., Biere, A.: Integrating Dependency schemes in search-based QBF solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 158–171. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14186-7_14
3. Lonsing, F., Biere, A.: A compact representation for syntactic dependencies in QBFs. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 398–411. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_37
4. Egly, U., Lonsing, F., Widl, M.: Long-distance resolution: proof generation and strategy extraction in search-based QBF solving. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 291–308. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_21
5. Giunchiglia, E., Narizzano, M., Pulina, L., Tacchella, A.: Quantified Boolean Formulas satisfiability library (QBFLIB) (2005). www.qbflib.org
6. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified Boolean formulas. J. Artif. Intell. Res. **26**, 371–416 (2006)
7. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_10
8. Janota, M., Marques-Silva, J.: 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 (2015)
9. Kaibel, V., Peinhardt, M.: On the bottleneck shortest path problem. Zib-report 06–22, Zuse Institute Berlin (2006)

10. Büning, H.K., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Information and Computation **17**(1), 12–18 (1995)
11. Peitl, T., Slivovsky, F., Szeider, S.: Dependency learning for QBF. J. Artif. Intell. Res., 65 (2019)
12. Peitl, T., Slivovsky, F., Szeider, S.: Long-distance Q-resolution with dependency schemes. J. Autom. Reason. **63**(1), 127–155 (2019)
13. Rabe, M.N., Tentrup, L.: CAQE: A certifying QBF solver. In: Kaivola, R., Wahl, T. (eds.) Formal Methods in Computer-Aided Design - FMCAD 2015, pp. 136–143. IEEE Computer Society (2015)
14. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. J. Autom. Reason. **42**(1), 77–97 (2009)
15. Slivovsky, F., Szeider, S.: Quantifier reordering for QBF. J. Autom. Reason. **56**(4), 459–477 (2016)
16. Slivovsky, F., Szeider, S.: Soundness of Q-resolution with dependency schemes. Theor. Comput. Sci. **612**, 83–101 (2016)
17. Gelder, A.: Variable Independence and resolution paths for quantified Boolean formulas. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 789–803. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_59
18. Wimmer, R., Reimer, S., Marin, P., Becker, B.: HQSpre – an effective preprocessor for QBF and DQBF. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 373–390. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_21
19. Zhang, L., Malik, S.: 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, 10–14 November 2002, pp. 442–449. ACM/IEEE Computer Society (2002)