Small Resolution Proofs for QBF using Dependency Treewidth

Eduard Eiben, Robert Ganian, and Sebastian Ordyniak

Algorithms and Complexity Group, TU Wien, Vienna, Austria
eduard.eiben@gmail.com, rganian@gmail.com, sordyniak@gmail.com

Abstract
In spite of the close connection between the evaluation of quantified Boolean formulas (QBF) and propositional satisfiability (SAT), tools and techniques which exploit structural properties of SAT instances are known to fail for QBF. This is especially true for the structural parameter treewidth, which has allowed the design of successful algorithms for SAT but cannot be straightforwardly applied to QBF since it does not take into account the interdependencies between quantified variables.

In this work we introduce and develop dependency treewidth, a new structural parameter based on treewidth which allows the efficient solution of QBF instances. Dependency treewidth pushes the frontiers of tractability for QBF by overcoming the limitations of previously introduced variants of treewidth for QBF. We augment our results by developing algorithms for computing the decompositions that are required to use the parameter.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The problem of evaluating quantified Boolean formulas (QBF) is a generalization of the propositional satisfiability problem (SAT) which naturally captures a range of computational tasks in areas such as verification, planning, knowledge representation and automated reasoning [11, 19, 23, 24]. QBF is the archetypical PSPACE-complete problem and is therefore believed to be computationally harder than NP-complete problems such as SAT [17, 20, 30].

In spite of the close connection between QBF and SAT, many of the tools and techniques which work for SAT are known not to help for QBF, and dynamic programming based on the structural parameter treewidth [2, 31] is perhaps the most prominent example of this behavior. Treewidth is a highly-established measure of how “treelike” an instance is, and in the SAT setting it is known that $n$-variable instances of treewidth at most $k$ can be solved in time at most $f(k) \cdot n^{O(1)}$ for a computable function $f$. Algorithms with running time in this form (i.e., $f(k) \cdot n^{O(1)}$, where $k$ is the parameter and the degree of the polynomial of $n$ is independent of $k$) are called fixed-parameter algorithms, and problems which admit such an algorithm (w.r.t. a certain parameter) belong to the class FPT. Furthermore, in the SAT setting, treewidth allows us to do more than merely solve the instance: it is also possible to find a so-called resolution proof [7, 5]. If the input was a non-instance, such a resolution proof contains additional information on “what makes it unsatisfiable” and hence can be more useful than outputting a mere Reject in practical settings.

In the QBF setting, the situation is considerably more complicated. It is known that QBF instances of bounded treewidth remain PSPACE-complete [2], and the intrinsic reason for this fact is that treewidth does not take into account the dependencies that arise between variables in QBF. So far, there have been several attempts at remedying this situation by introducing variants of treewidth which support fixed-parameter algorithms for QBF: prefix pathwidth (along with prefix treewidth) [12] and respectful treewidth [2], along with two other...
parameters [11, 6] which originate from a different setting but can also be adapted to obtain fixed-parameter algorithms for QBF. We refer to Subsection 3.2 for a comparison of these parameters. Aside from algorithms with runtime guarantees, it is worth noting that empirical connections between treewidth and QBF have also been studied in the literature [21, 22].

In this work we introduce and develop dependency treewidth, a new structural parameter based on treewidth which supports fixed-parameter algorithms for QBF. Dependency treewidth pushes the frontiers of tractability for QBF by overcoming the limitations of both the previously introduced prefix and respectful variants. Compared to the former, this new parameter allows the computation of resolution proofs analogous to the case of classical treewidth for SAT instances. Prefix pathwidth relies on entirely different techniques to solve QBF and does not yield small resolution proofs. Moreover, the running time of the fixed-parameter algorithm which uses prefix pathwidth has a triple-exponential dependency on the parameter \( k \), while dependency treewidth supports a considerably more efficient \( O(3^{2k} nk) \)-time algorithm for QBF.

Unlike respectful treewidth and its variants, which only take the basic dependencies between variables into account, dependency treewidth can be used in conjunction with the so-called dependency schemes introduced by Samer and Szeider [25, 27], see also the work of Biere and Lonsing [3]. Dependency schemes allow an in-depth analysis of how the assignment of individual variables in a QBF depends on other variables, and research in this direction has uncovered a large number of distinct dependency schemes with varying complexities. The most basic dependency scheme is called the trivial dependency scheme [25], which stipulates that each variable depends on all variables with distinct quantification which precede it in the prefix. Respectful treewidth in fact coincides with dependency treewidth when the trivial dependency scheme is used, but more advanced dependency schemes allow us to efficiently solve instances which otherwise remain out of the reach of state-of-the-art techniques.

Crucially, all of the structural parameters mentioned above require a so-called decomposition in order to solve QBF; computing these decompositions is typically an NP-hard problem. A large part of our technical contribution lies in developing algorithms to compute decompositions for dependency treewidth. Without such algorithms, it would not be possible to use the parameter unless a decomposition were supplied as part of the input (an unrealistic assumption in practical settings). It is worth noting that all of these algorithms can also be used to find respectful tree decompositions, where the question of finding suitable decompositions was left open [2]. We provide two algorithms for computing dependency tree decompositions, each suitable for use under different situations.

The article is structured as follows. After the preliminaries, we introduce the parameter and show how to use it to solve QBF (assuming a decomposition has already been computed). This section also contains an in-depth overview and comparison of previous work in the area. A separate section then introduces other equivalent characterizations of dependency treewidth. The last technical section contains our algorithms for finding dependency tree decompositions, after which we provide concluding notes and remarks.

## 2 Preliminaries

For \( i \in \mathbb{N} \), we let \([i]\) denote the set \( \{1, \ldots, i\} \). We refer to the book by Diestel [8] for standard graph terminology. Given a graph \( G \), we denote by \( V(G) \) and \( E(G) \) its vertex and edge set, respectively. We use \( ab \) as a shorthand for the edge \( \{a, b\} \). For \( V' \subseteq V(G) \), the guards of \( V' \) (denoted \( \delta(V') \)) are the vertices in \( V(G) \setminus V' \) with at least one neighbor in \( V' \).

We refer to the standard textbooks [10, 14] for an in-depth overview of parameterized
complexity theory. Here, we only recall that a parameterized problem \((Q, \kappa)\) is a problem \(Q \subseteq \Sigma^*\) together with a parameterization \(\kappa: \Sigma^* \rightarrow \mathbb{N}\), where \(\Sigma\) is a finite alphabet. A parameterized problem \((Q, \kappa)\) is fixed-parameter tractable (w.r.t. \(\kappa\)), in short FPT, if there exists a decision algorithm for \(Q\), a computable function \(f: \mathbb{N} \rightarrow \mathbb{N}\), and a polynomial function \(p: \mathbb{N} \rightarrow \mathbb{N}\), such that for all \(x \in \Sigma^*\), the running time of the algorithm on \(x\) is at most \(f(\kappa(x)) \cdot p(|x|)\). Algorithms with this running time are called fixed-parameter algorithms.

### 2.1 Quantified Boolean Formulas

For a set of propositional variables \(K\), a literal is either a variable \(x \in K\) or its negation \(\bar{x}\). A clause is a disjunction over literals. A propositional formula in conjunctive normal form (i.e., a CNF formula) is a conjunction over clauses. Given a CNF formula \(\phi\), we denote the set of variables which occur in \(\phi\) by \(\text{var}(\phi)\). For notational purposes, we will view a clause as a set of literals and a CNF formula as a set of clauses.

A quantified Boolean formula is a tuple \((\phi, \tau)\) where \(\phi\) is a CNF formula and \(\tau\) is a sequence of quantified variables, denoted \(\text{var}(\tau)\), which satisfies \(\text{var}(\tau) \supseteq \text{var}(\phi)\); then \(\phi\) is called the matrix and \(\tau\) is called the prefix. A QBF \((\phi, \tau)\) is true if the formula \(\tau \phi\) is true. A quantifier block is a maximal sequence of consecutive variables with the same quantifier. An assignment is a mapping from (a subset of) the variables to \(\{0, 1\}\).

The primal graph of a QBF \(I = (\phi, \tau)\) is the graph \(G_I\) defined as follows. The vertex set of \(G_I\) consists of every variable which occurs in \(\phi\), and \(st\) is an edge in \(G_I\) if there exists a clause in \(\phi\) containing both \(s\) and \(t\).

### 2.2 Dependency Posets for QBF

Before proceeding, we define a few standard notions related to posets which will be used throughout the paper. A partially ordered set (poset) \(\mathcal{V}\) is a pair \((V, \leq^V)\) where \(V\) is a set and \(\leq^V\) is a reflexive, antisymmetric, and transitive binary relation over \(V\). A chain \(W\) of \(\mathcal{V}\) is a subset of \(V\) such that \(x \leq^V y\) or \(y \leq^V x\) for every \(x, y \in W\). A chain partition of \(\mathcal{V}\) is a tuple \((W_1, \ldots, W_k)\) such that \(\{W_1, \ldots, W_k\}\) is a partition of \(V\) and for every \(i\) with \(1 \leq i \leq k\) the poset induced by \(W_i\) is a chain of \(\mathcal{V}\). An anti-chain \(A\) of \(\mathcal{V}\) is a subset of \(V\) such that for all \(x, y \in A\) neither \(x \leq^V y\) nor \(y \leq^V x\). The width (or poset-width) of a poset \(\mathcal{V}\), denoted by \(\text{width}(\mathcal{V})\), is the maximum cardinality of any anti-chain of \(\mathcal{V}\). A poset of width 1 is called a linear order. A linear extension of a poset \(P = (P, \leq^P)\) is a relation \(\preceq^P\) over \(P\) such that \(x \preceq^P y\) whenever \(x \leq^P y\) and the poset \(P^\ast = (P, \preceq^P)\) is a linear order. A subset \(A\) of \(V\) is downward-closed if for every \(a \in A\) it holds that \(b \leq^V a \implies b \in A\). A reverse of a poset is obtained by reversing each relation in the poset. For brevity we will often write \(\leq^V\) to refer to the poset \(\mathcal{V} := (V, \leq^V)\).

**Proposition 1 ([13]).** Let \(\mathcal{V}\) be a poset. Then in time \(O(\text{width}(\mathcal{V}) \cdot ||\mathcal{V}||^2)\), it is possible to compute both \(\text{width}(\mathcal{V}) = w\) and a corresponding chain partition \((W_1, \ldots, W_w)\) of \(\mathcal{V}\).

We use dependency posets to provide a general and formal way of speaking about the various dependency schemes introduced for QBF [25]. It is important to note that dependency schemes in general are too broad a notion for our purposes; for instance, it is known that some dependency schemes do not even give rise to sound resolution proof systems. Here we focus solely on so-called permutation dependency schemes [25], which is a general class containing all commonly used dependency schemes that give rise to sound resolution proof
systems. This leads us to our definition of dependency posets, which allow us to capture all permutation dependency schemes.

Given a QBF $I = (\phi, \tau)$, a dependency poset $V = (\text{var}(\phi), \leq^I)$ of $I$ is a poset over $\text{var}(\phi)$ with the following properties:

1. for all $x, y \in \text{var}(\phi)$, if $x \leq^I y$, then $x$ is before $y$ in the prefix, and
2. given any linear extension $\preceq$ of $V$, the QBF $I' = (\phi, \tau_{\preceq})$, obtained by permutation of the prefix $\tau$ according the $\preceq$, is true iff $I$ is true.

The trivial dependency scheme is one specific example of a permutation dependency scheme. This gives rise to the trivial dependency poset, which sets $x \leq y$ whenever $x, y$ are in different quantifier blocks and $x$ is before $y$ in the prefix. However, more refined permutation dependency schemes which give rise to other dependency posets are known to exist and can be computed efficiently \cite{25, 28}. In particular, it is easy to verify that a dependency poset can be computed from any permutation dependency scheme in polynomial time (by computing the transitive closure).

To illustrate these definitions, consider the following QBF:

$$\exists a \forall b \exists c (a \lor c) \land (b \lor c)$$

Then the trivial dependency poset would set $a \leq b \leq c$. However, for instance the resolution path dependency poset (arising from the resolution path dependency scheme \cite{32, 26}) contains a single relation $b \leq c$ (in this case, $a$ is incomparable to both $b$ and $c$).

### 2.3 Q-resolution

Q-resolution is a sound and complete resolution system for QBF \cite{16}. Our goal here is to formalize the required steps for the Davis Putnam variant of Q-resolution.

We begin with a bit of required notation. For a QBF $I = (\phi, \tau)$ and a variable $x \in \text{var}(\phi)$, let $\phi_x$ be the set of all clauses in $\phi$ containing the literal $x$ and similarly let $\phi_{\bar{x}}$ be the set of all clauses containing literal $\bar{x}$. We denote by $res(I, x)$ the QBF $I' = (\phi', \tau')$ such that $\tau' = \tau \setminus \{x\}$ and $\phi' = \phi \setminus (\phi_x \cup \phi_{\bar{x}}) \cup \{(D \setminus \{x\}) \cup (C \setminus \{\bar{x}\}) | D \in \phi_x; C \in \phi_{\bar{x}}\}$; informally, the two clause-sets are pairwise merged to create new clauses which do not contain $x$. For a QBF $I = (\phi, \tau)$ and a variable $x \in \text{var}(\phi)$ we denote by $I \setminus x$ the QBF $I = (\phi', \tau \setminus \{x\})$, where we get $\phi'$ from $\phi$ by removing all occurrences of $x$ and $\bar{x}$.

**Lemma 2.** Let $I = (\phi, \tau)$ and $x \in \text{var}(\phi)$ be the last variable in $\tau$. If $x$ is existentially quantified, then $I$ is true if and only if $res(I, x)$ is true.

**Proof.** Assume $I$ is true and let $F$ be a winning strategy for existential player in $I$. We will show that $F$ is also a winning existential strategy in $res(I, x)$. Assume that the existential player played according to $F$ in $res(I, x)$, but there is a clause $B$ that is not satisfied at the end of the game. Clearly $B \in \{(D \setminus \{x\}) \cup (C \setminus \{\bar{x}\}) \cup (D \setminus \{\bar{x}\}) \cup (C \setminus \{x\}) | D \in \phi_x; C \in \phi_{\bar{x}}\}$, otherwise $B$ is also a clause of $I$ and hence it has to be satisfied due to the existential player using $F$. In particular, $B = (D \setminus \{x\}) \cup (C \setminus \{\bar{x}\})$ for some $D \in \phi_x$ and $C \in \phi_{\bar{x}}$. Now if $F$ assigns $x$ to 1, since $F$ is a winning strategy it follows that $C$ must be satisfied by some other literal, and hence $B$ must also be satisfied—a contradiction. A symmetric argument also leads to a contradiction if $F$ assigns $x$ to 0.

Assume now that $res(I, x)$ is true and let $F$ be a winning strategy for the existential player in $res(I, x)$. Now suppose that after all variables of $res(I, x)$ have been assigned according to the strategy $F$, there is some $D \in \phi_x$ such that $D \setminus \{x\}$ is false. Since $(D \setminus \{x\}) \cup (C \setminus \{\bar{x}\})$ is true for all $C \in \phi_{\bar{x}}$, it means that all $C \in \phi_{\bar{x}}$ are true before we
assign \( x \), and our strategy can assign \( x \) to 1. On the other hand if \( D \setminus \{ x \} \) is true for all \( D \) in \( \phi_x \), our strategy assigns \( x \) to 0 and again satisfies all clauses of \( I \).

\[ \text{Lemma 3. Let } I = (\phi, \tau) \text{ and } x \in \text{var}(\phi) \text{ be the last variable in } \tau. \text{ If } x \text{ is universally quantified, then } I \text{ is true if and only if } I \setminus x \text{ is true.} \]

\[ \text{Proof. We will prove an equivalent statement: } I \setminus x \text{ is false if and only if } I \text{ is false. It is easy to see that if } \mathcal{F} \text{ is a winning strategy for the universal player in } I, \text{ then if he plays according } \mathcal{F}, \text{ then when the universal should assign the last variable } x \text{ there is either a clause that is already false and does not contain } x, \text{ or a clause that contains } x \text{ and is false after an assignment of } x \text{ according to } \mathcal{F}. \text{ In both cases } I \setminus x \text{ contains a clause that is false.} \]

On the other hand, assume \( \mathcal{F} \) is a winning strategy for the universal player in \( I \setminus x \) and the universal plays according to \( \mathcal{F} \) in \( I \) until all variables but \( x \) are assigned. Clearly, this strategy leads to an assignment of variables such that there is a clause \( B \) in \( I \setminus x \), which is false under this assignment. From the definition of \( I \setminus x \), it is easy to observe that \( I \) contains a clause \( B' \), which is equal to one of the following: \( B, B \cup \{ x \}, \text{ or } B \cup \{ \bar{x} \} \). It is straightforward to extend \( \mathcal{F} \) in a way that whenever \( \mathcal{F} \) falsifies a clause \( B \), then the new strategy falsifies the clause \( B' \).

\[ \] 2.4 Treewidth

Here we will introduce three standard characterizations of treewidth [15]: tree decompositions, elimination orderings, and cops and robber games. These will play an important role later on, when we define their counterparts in the dependency treewidth setting and use these in our algorithms.

**Tree decomposition:** A tree decomposition of a graph \( G \) is a pair \((T, \chi)\), where \( T \) is a rooted tree and \( \chi \) is a function from \( V(T) \) to subsets of \( V(G) \), called a bag, such that the following properties hold: (T1) \( \bigcup_{t \in V(T)} \chi(t) = V(G) \), (T2) for each \( uv \in E(G) \) there exists \( t \in V(T) \) such that \( u, v \in \chi(t) \), and (T3) for every \( u \in V(G) \), the set \( T_u = \{ t \in V(T) : u \in \chi(t) \} \) induces a connected subtree of \( T \).

To distinguish between the vertices of the tree \( T \) and the vertices of the graph \( G \), we will refer to the vertices of \( T \) as nodes. The width of the tree decomposition \( T \) is \( \max_{t \in T} |\chi(t)| - 1 \). The treewidth of \( G \), \( \text{tw}(G) \), is the minimum width over all tree decompositions of \( G \). For a node \( t \in V(T) \), we denote by \( T_t \) the subtree of \( T \) rooted at \( t \). The following fact will be useful later on:

\[ \textbf{Proposition 4. Let } T = (T, \chi) \text{ be a tree decomposition of a graph } G \text{ and } t \in V(T) \text{ a node with parent } p \text{ in } T. \text{ Then } \chi(p) \cap \chi(t) \text{ separates } \chi(T_t) \setminus \chi(p) \text{ from the rest of } G. \]

**Elimination ordering:** An elimination ordering of a graph is a linear order of its vertices. Given an elimination ordering \( \phi \) of the graph \( G \), the fill-in graph \( H \) of \( G \) w.r.t. \( \phi \) is the unique minimal graph such that:

\[ V(G) = V(H). \]
\[ E(H) \supseteq E(G). \]
\[ \text{If } 0 \leq k < i < j \leq n \text{ and } v_i, v_j \in N_H(v_k), \text{ then } v_iv_j \in E(H). \]

The width of elimination ordering \( \phi \) is the maximum number of neighbors of any vertex \( v \) that are larger than \( v \) (w.r.t. \( \phi \)) in \( H \).

**Monotone** cops and robber game: The cops and robber game is played between two players (the cop-player and the robber-player) on a graph \( G \). A position in the game is a pair \((C, R)\) where \( C \subseteq V(G) \) is the position of the cop-player and \( R \) is a (possibly empty)
connected component of $G \setminus C$ representing the position of the robber-player. A move from position $(C, R)$ to position $(C', R')$ is legal if it satisfies the following conditions:

1. $R$ and $R'$ are contained in the same component of $G \setminus (C \cap C')$,
2. $\delta(R) \subseteq C'$.

A play $P$ is a sequence $((\emptyset, R_0),..., (C_n, R_n))$ of positions such that for every $i$ with $1 \leq i < n$ it holds that the move from $(C_i, R_i)$ to $(C_{i+1}, R_{i+1})$ is legal; the cop-number of a play is $\max_{1 \leq i \leq n} |C_i|$. A play $P$ is won by the cop-player if $R_n = \emptyset$, otherwise it is won by the robber-player. The cop-number of a strategy for the cop player is maximum cop-number over all plays that can arise from this strategy. Finally, the cop-number of $G$ is the minimum cop-number of a winning strategy for the cop player.

\[ \textbf{Proposition 5} \text{ ([18])} \]
Let $G$ be a graph. The following three claims are equivalent:

- $G$ has treewidth $k$,
- $G$ has an elimination ordering of width $k$,
- $G$ has cop-number $k$.

## 3 Dependency Treewidth for QBF

We are now ready to introduce our parameter. We remark that in the case of dependency treewidth, it is advantageous to start with a counterpart to the elimination ordering characterization of classical treewidth, as this is used extensively in our algorithm for solving QBF. We provide other equivalent characterizations of dependency treewidth (representing the counterparts to tree decompositions and cops and robber games) in Section 4; these are not only theoretically interesting, but serve an important role in our algorithms for computing the dependency treewidth.

Let $I = (\phi, \tau)$ be a QBF instance with a dependency poset $\mathcal{P}$. An elimination ordering of $G_I$ is compatible with $\mathcal{P}$ if it is a linear extension of the reverse of $\mathcal{P}$; intuitively, this corresponds to being forced to eliminate variables that have the most dependencies first. For instance, if $\mathcal{P}$ is a trivial dependency poset then a compatible elimination ordering must begin eliminating from the rightmost block of the prefix. We call an elimination ordering of $G_I$ that is compatible with $\mathcal{P}$ a $\mathcal{P}$-elimination ordering (or dependency elimination ordering). The dependency treewidth w.r.t. $\mathcal{P}$ is then the minimum width of a $\mathcal{P}$-elimination ordering.

### 3.1 Using dependency treewidth

Our first task is to show how dependency elimination orderings can be used to solve QBF.

\[ \textbf{Theorem 6} \text{.} \] There is an algorithm that given

1. a QBF $I$ with $n$ variables and $m$ clauses,
2. a dependency poset $\mathcal{P}$ for $I$, and
3. a $\mathcal{P}$-elimination ordering $\pi$ of width $k$,

decides whether $I$ is true in time $O(3^k n)$. Moreover, if $I$ is false, then the algorithm outputs a $Q$-resolution refutation of size $O(3^k n)$.

\[ \textbf{Proof} \text{.} \] Let $I = (\phi, \tau)$ and let $x_1, ..., x_n$ denote the variables of $\phi$ such that $x_i \leq \pi x_{i+1}$ for all $1 \leq i < n$. From the definition of the dependency poset and the fact that $\pi$ is a dependency elimination ordering, it follows that the QBF instance $I' = (\phi, \tau')$, where $\tau'$ is the reverse of $\pi$, is true if and only if $I$ is true.
To solve $I$ we use a modification of the Davis Putnam resolution algorithm [7]. We start with instance $I'$ and recursively eliminate the last variable in the prefix using Lemmas 2 and 3 until we either run out of variables or we introduce as a resolvent a non-tautological clause that is either empty or contains only universally quantified variables. We show that each variable we eliminate has the property that it only shares clauses with at most $k$ other variables, and in this case we introduce at most $3^k$ clauses of size at most $k$ at each step.

From now on let $H$ be the fill-in graph of the primal graph of $I$ with respect to $\pi$, and let us define $I_i = (\phi^i, \tau^i)$ for $1 \leq i \leq n$ as follows:

1. $I_1 = I'$,
2. $I_{i+1} = I_i \setminus x_i$ if $x_i$ is universally quantified, and
3. $I_{i+1} = \text{res}(I_i, x_i)$, if $x_i$ is existentially quantified.

Note that $x_i$ is always the last variable of the prefix of $I_i$ and it follows from Lemmas 2 and 3 that $I_{i+1}$ is true if and only if $I_i$ is true. Moreover, $I_n$ only contains a single variable, and hence can be decided in constant time. In the following, we show by induction that $I_{i+1}$ contains at most $3^k$ new clauses, i.e., clauses not contained in $I_i$. To this end, we show and use the fact that both $\phi^i_2$ and $\phi^i_3$ contain at most $3^k$ clauses, and this is sufficient to ensure a small Q-resolution refutation if the instance is false.

\textbf{Claim 1.} The instance $I_{i+1}$ contains at most $3^k$ clauses not contained in $I_i$. Furthermore, if the primal graph of $I_i$ is a subgraph of $H$, then so is the primal graph of $I_{i+1}$.

\textbf{Proof of the Claim.} We distinguish two cases: either $x_i$ is universal, or existential. In the former case, it is easily observed that the primal graph of $I_i \setminus x_i$ is a subgraph of the primal graph of $I_i$ obtained by removing $x_i$ from the graph. Hence, in this case the primal graph of $I_{i+1}$ is a subgraph of $H$ as well. Moreover, we do not add any new clauses, only remove $x_i$ from already existing ones.

In the later case, let $x_i$ be an existentially quantified variable. Let $x_px_q$ be an edge in the primal graph of $\text{res}(I_i, x_i)$, but not in the primal graph of $I_i$. Clearly, $x_p$ and $x_q$ are in a newly added clause $B = (D \setminus \{x\}) \cup (C \setminus \{\bar{x}\})$ such that $D \in \phi^i_2$ and $C \in \phi^i_3$. But that means that both $x_p$ and $x_q$ are in a clause with $x_i$ in $I_i$. Hence, $x_px_i$ and $x_qx_i$ are both edges in the primal graph of $I_i$ and also in $H$. But, $p > i$ and $q > i$, since otherwise $x_p$ and $x_q$ will not appear in $I_{i+1}$. Since $H$ is the fill-in graph w.r.t. $\pi$, $H$ contains $x_px_q$ as well and $H$ is a supergraph of the primal graph of $I_i$. Moreover, as $\pi$ is an elimination ordering of the primal graph of $I$ of width $k$, there are at most $k$ variables in $\phi^i$ that appear with $x_i$ in a clause in $I_i$. Hence there are at most $3^k$ different clauses containing these variables, and so $\phi^i_2$, $\phi^i_3$, contain at most $3^k$ clauses of size at most $k + 1$. Finally, the set of new clauses $\{(D \setminus \{x_i\}) \cup (C \setminus \{\bar{x}_i\})|D \in \phi^i_2, C \in \phi^i_3\}$ contains at most $3^k$ clauses too, and the claim follows.

Since $I$ is equivalent to $I_n$, it is easy to see that if $I$ is false, then $I_n$ either contains the empty clause, or two clauses $\{x_n\}$ and $\{\bar{x}_n\}$, or $x_n$ is universally quantified. In these cases it is easy to see that the union of clauses of all $I_i$ (including the empty clause, if it is not already in $I_n$) is a Q-resolution refutation. Moreover, it follows from Claim 1 that this Q-resolution refutation has at most $m + 3^k n$ clauses.

For the runtime analysis, the time necessary to compute a variable-clause adjacency list data structure (which will be useful for the next step) is upper-bounded by $O(mk)$ due to clauses having size at most $k$ (since the width of the elimination ordering is at most $k$). If $x$ is universal, then $I \setminus x$ can be computed in time $O(3^k)$, since $x$ occurs in at most $3^k$ clauses. If $x$ is existential, then we need to compute the new clauses which takes time at
most $O(3^k \cdot k)$ since there are at most $3^k$ pairs of clauses containing $x$ and each such clause has size at most $k$.

Finally, observe that the total number of clauses cannot exceed $n \cdot k$ (because the treewidth of the matrix is also bounded by $k$), and so both $O(mk)$ and $m$ are superseded by the other term in the runtime and the resolution refutation size.

### 3.2 A Comparison of Decompositional Parameters for QBF

As was mentioned in the introduction, two dedicated decompositional parameters have previously been introduced specifically for evaluating quantified Boolean formulas: prefix pathwidth (and, more generally, prefix treewidth) [12] and respectful treewidth [2]. The first task of this section is to outline the advantages of dependency treewidth compared to these two parameters.

Prefix pathwidth is based on bounding the number of viable strategies in the classical two-player game characterization of the QBF problem [12]. As such, it decomposes the dependency structure of a QBF instance beginning from variables that have the least dependencies (i.e., may appear earlier in the prefix). On the other hand, our dependency treewidth is based on Q-resolution and thus decomposes the dependency structure beginning from variables that have the most dependencies (i.e., may appear last in the prefix). Lemma 7 shows that both approaches are, in principle, incomparable. That being said, dependency treewidth has two critical advantages over prefix treewidth/pathwidth:

1. dependency treewidth outputs small resolution proofs, while it is not at all clear whether the latter can be used to obtain such resolution proofs;
2. dependency treewidth supports a single-exponential fixed-parameter algorithm for QBF (Theorem 6), while the latter uses a prohibitive triple-exponential algorithm [12].

▶ Lemma 7. Let us fix the trivial dependency poset. There exist infinite classes $A, B$ of QBF instances such that:

a. $A$ has unbounded dependency treewidth but prefix pathwidth at most 1;

b. $B$ has unbounded prefix pathwidth (and prefix treewidth) but dependency treewidth at most 1.

Proof. a. Let

$$A_i = \exists x_1, \ldots, x_i \forall y \exists x (y \lor x) \land \bigwedge_{j=1}^{i} (x_j \lor x).$$

The trivial dependency poset $P_i$ for $A_i$ would be $\{x_1, \ldots, x_i\} \preceq y \preceq x$. Hence every $P_i$-elimination ordering must start with $x$, and then the width of such an ordering would be $i+1$. On the other hand, one can observe [12] that the path decomposition $Q = (Q_1, \ldots, Q_{i+1})$, where $Q_j = \{x_j, x\}$ for $1 \leq j \leq i$ and $Q_{i+1} = \{y, x\}$, is a prefix path-decomposition w.r.t. $P_i$ of width 1.

b. Consider the following formula with alternating prefix:

$$B_i = \exists x_1 \forall x_2 \ldots \forall x_{2^i} \exists x_{2^i+1} \bigwedge_{j=1}^{2^i-1} ((x_j \lor x_{2j}) \land (x_j \lor x_{2j+1})).$$

Since the quantifiers in the prefix of $B_i$ alternate, the trivial dependency poset $P_i$ for $B_i$ would be the linear order $x_1 \leq x_2 \leq \cdots \leq x_{2^i}$. It is readily observed that the primal graph of $B_i$ is a balanced binary tree of depth $i$, and it is known that the pathwidth of such trees is
From the fact that pathwidth is a trivial lower bound for prefix pathwidth together with previous work on prefix treewidth \[12, \text{Theorem 6}\], it follows that \(i - 1\) is a lower bound on the prefix treewidth of \(B_i\).

On the other hand, since \(P_i\) is linear order, the only elimination ordering compatible with \(P_i\) is the reverse of \(P_i\). Moreover, from the definition of \(B_i\), it is easily seen that \(x_j\) has at most 1 neighbor that is smaller w.r.t. \(P_i\), namely \(x_{\lfloor j/2 \rfloor}\). Therefore, the dependency treewidth of \(B_i\) is 1.

Respectful treewidth coincides with dependency treewidth when the trivial dependency scheme is used, i.e., represents a special case of our measure. Unsurprisingly, the use of more advanced dependency schemes (such as the resolution path dependency scheme \[32, 28\]) allows the successful deployment of dependency treewidth on much more general classes of QBF instances. Furthermore, dependency treewidth with such dependency schemes will always be upper-bounded by respectful treewidth, and so algorithms based on dependency treewidth will outperform the previously introduced respectful treewidth based algorithms.

**Lemma 8.** There exists an infinite class \(C\) of QBF instances such that \(C\) has unbounded dependency treewidth with respect to the trivial dependency poset but dependency treewidth at most 1 with respect to the resolution-path dependency poset.

**Proof.** Recall the previous example:

\[
A_i = \exists x_1, \ldots, x_i \forall y \exists x (y \lor x) \land \bigwedge_{j=1}^{i} (x_j \lor x).
\]

We have already established that \(A_i\) has dependency treewidth \(i + 1\) when the trivial dependency poset is used. However, e.g., the resolution-path dependency poset \[32, 26\] contains a single relation \(y \leq x\). Since the primal graph of \(A_i\) is a star, it is then easy to verify that \(A_i\) has dependency treewidth 1 when the resolution-path dependency poset is used.

Finally, we note that the idea of exploiting dependencies among variables has also given rise to similarly flavored structural measures in the areas of first-order model checking (first order treewidth) \[1\] and quantified constraint satisfaction (CD-width) \[6\]. Even though the settings differ, Theorem 5.5 \[1\] and Theorem 5.1 \[6\] can both be translated to a basic variant of Theorem 6. We note that this readily-obtained variant of Theorem \[6\] would not account for dependency schemes. We conclude this subsection with two lemmas which show that there are classes of QBF instances that can be handled by our approach but are not covered by the results of Adler, Weyer \[1\] and Chen, Dalmau \[6\].

Before we compare the dependency treewidth with the CD-width of Chen and Dalmau, we will first define their parameter (or, more specifically, provide a translation into the QBF setting). CD-width is also based on an elimination ordering, however with a slight modification: they also use the fact that we can eliminate universal variables without introducing new clauses (see Lemma \[3\]) and therefore we can eliminate the last universal variable even if it appears together with many variables. Formally the elimination ordering by Chen and Dalmau is defined as follows:

---

\[1\] We remark that in their paper, the authors refer to their parameter simply as “the width”. For disambiguation, here we call it CD-width (shorthand for Chen-Dalmau’s width).
Definition 9. Let \( I = (\phi, \tau) \) be a QBF instance. Given a linear ordering \( \preceq \) of \( \text{var}(\phi) \), the CD-fill-in graph \( H_{\preceq} \) of \( I \) is the unique minimal graph such that:

- \( V(G_I) = V(H_{\preceq}) \).
- \( E(H_{\preceq}) \supseteq E(G_I) \).
- If \( u \preceq v \preceq w \), \( u \) is existentially quantified, and \( v, w \in N_{H_{\preceq}}(u) \), then \( vw \in E(H_{\preceq}) \).

A CD-elimination ordering of a QBF instance \( I = (\phi, \tau) \) is a linear order \( \preceq \) of \( \text{var}(\phi) \) such that for each existentially quantified \( x \) and each universally quantified \( y \) in \( \text{var}(\phi) \):

- If \( y \) is before \( x \) in \( \tau \), then \( x \preceq y \).
- If \( x \) is before \( y \) in \( \tau \) and there is an edge \( xy \) in \( H_{\preceq} \), then \( y \preceq x \).

The width of a CD-elimination ordering \( \preceq \) is the maximum number of neighbors of an existentially quantified vertex \( v \) that are larger than \( v \) (w.r.t. \( \preceq \)) in \( H_{\preceq} \).

We remark that one could also use the less restrictive form of elimination orderings considered above with any dependency poset, obtaining a more general variant of Theorem 6. However, such a notion would lose many of the nice structural properties used in our algorithms for finding the decompositions; for instance, the result is no longer a restriction of treewidth and does not have any immediate cops-and-robber game characterization. Hence finding such an ordering of small width would become a more challenging problem.

Lemma 10. There exist infinite classes \( \mathcal{D}, \mathcal{E} \) of QBF instances such that:

a. \( \mathcal{D} \) has unbounded CD-width but dependency treewidth at most 1 w.r.t. the resolution-path dependency poset.

b. \( \mathcal{E} \) has unbounded dependency treewidth w.r.t. any dependency poset but CD-width at most 1.

Proof. a. Recall the previous example:

\[
A_i = \exists x_1, \ldots, x_i \forall y \exists x (y \lor x) \land \bigwedge_{j=1}^{i} (x_j \lor x).
\]

We have already established that \( A_i \) has dependency treewidth 1 when the resolution-path dependency poset is used. To establish the other directions it suffices to observe that Example 3.6 of Chen and Dalmau \cite{ChenDalmau2007} uses the class of prefixed graphs obtained from \( \mathcal{A} \) by taking primal graph of instances \( A_i \) and keeping the same prefix as an example of class with unbounded elimination width.

b. Let

\[
E_i = \forall x_1 \forall x_2 \ldots \forall x_i \bigwedge_{1 \leq p < q \leq i} (x_p \lor x_q).
\]

As \( E_i \) does not contain any existentially quantified variable, it is easy to observe that CD-width of \( E_i \) is 0. However, the primal graph of \( E_i \) is a clique and hence dependency treewidth is \( i - 1 \) w.r.t. every possible dependency poset.

Similarly as above, before we compare first order treewidth to dependency treewidth, we give some necessary definitions.

Let \( \prec \) be a binary relation on the variables of some QBF instance \( I = (\phi, \tau) \). Then two variables \( x \) and \( y \) are entangled with respect to \( \prec \) and \( I \), if \( x \) occurs in a clause with some variable \( z \) such that \( y \prec z \) and \( y \) occurs in a clause with some variable \( z \) such that \( x \prec z \).

Definition 11. Let \( I = (\phi, \tau) \) be a QBF instance. Then \( \succeq_{\text{FO}} \) is the minimal (with respect to \( \subseteq \)) binary relation on \( \text{var}(\phi) \), such that the following holds:
(1) $\preceq^{\text{FO}}$ is reflexive.
(2) $\preceq^{\text{FO}}$ is transitive.
(3) If $x$ is before $y$ in $\tau$, $x$ and $y$ have different quantifiers in $\tau$, and there is a sequence $x = z_0, \ldots, z_n = y$ of variables such that for all $0 \leq i < n$ we have that $z_i$ and $z_{i+1}$ are entangled w.r.t. $\preceq_I$ and $I$ and that $x \preceq_I z_i$ or $y \preceq^{\text{FO}}_I z_i$, then $x \preceq^{\text{FO}}_I y$.

\begin{definition}
Let $I = (\phi, \tau)$ be a QBF instance and $x \in \var(\phi)$. The essential alternation depth of $x$ in $I$, denoted by $\operatorname{ead}(x)$, is the maximum over all $\preceq^{\text{FO}}_I$-paths $P$ ending in $x$ of the number of quantifier changes in $P$, adding +1 in case the first variable on $P$ is existentially quantified and +2 if it is universally quantified.

For a QBF instance $I = (\phi, \tau)$, let $\varphi = (\var(\phi), \preceq^{\text{FO}}_I)$ be the poset such that $x \preceq^{\text{FO}}_I y$ if and only if $\operatorname{ead}(x) \leq \operatorname{ead}(y)$. Notice that $x \preceq^{\text{FO}}_I y$ implies $\operatorname{ead}(x) \leq \operatorname{ead}(y)$ and hence $\varphi$ is an extension of $\preceq^{\text{FO}}_I$. The first order treewidth is the minimal width of an elimination ordering that is compatible with $\varphi$.

Now we are ready to prove the following lemma.

\begin{lemma}
There exists an infinite class $\mathcal{F}$ of QBF instances such that $\mathcal{F}$ has unbounded first order treewidth but dependency treewidth at most 2 with respect to the resolution-path dependency poset.
\end{lemma}

\begin{proof}
Let $F_i = \forall x_1 \forall y_1 \exists x_2 \cdots \exists x_{2i} \forall y_2 \exists x_1 y_1 \forall z$

$$
(z \lor x_1 \lor y_1) \bigwedge_{j=1}^{2i-1} [(x_j \lor x_{j+1}) \land (y_j \lor y_{j+1})] \land \bigwedge_{j=1}^{i} (x_{2j-1} \lor y_1).
$$

The resolution-path dependency poset would give us following relations: a chain $y_{2i} \leq \cdots \leq y_2 \leq y_1 \leq z$ and a chain $x_{2i} \leq \cdots \leq x_2 \leq x_1 \leq z$. It is readily observed that the elimination ordering $zx_1x_2 \cdots x_{2i}y_1y_2 \cdots y_2$, with width 2 is compatible with this dependency poset.

On the other hand, one can observe that $x_{2j} \preceq^{\text{FO}}_{F_i} x_{2j-1}$ and that $y_1$ and $x_{2j-1}$ are entangled w.r.t. $\preceq^{\text{FO}}_{F_i}$ and hence also $x_2 \preceq^{\text{FO}}_{F_i} y_1$ for all $1 \leq j \leq i$. It follows that any elimination ordering that is compatible with $\varphi$ must have $y_1$ before all $x_j$ for $2 \leq j \leq 2i$ and the width of such elimination ordering is at least $i - 1$.

Finally, we remark that if one were to show that $\varphi$ is a dependency poset, then this would imply that first order treewidth is a special case of dependency treewidth (for this dependency poset). However, proving such a claim goes beyond the scope of this paper.

\section{Dependency Treewidth: Characterizations}

In this section we obtain other equivalent characterizations of dependency treewidth. The purpose of this endeavor is twofold. From a theoretical standpoint, having several natural characterizations (corresponding to the characterizations of treewidth) is not only interesting but also, in some sense, highlights the solid foundations of a structural parameter. From a practical standpoint, the presented characterizations play an important role in Section 5 which is devoted to algorithms for finding optimal dependency elimination orderings.

\textbf{Dependency tree decomposition}: Let $I$ be a QBF instance with primal graph $G$ and dependency poset $\varphi$ and let $(T, \chi)$ be a tree decomposition of $G$. Note that the rooted tree...
T naturally induces a partial order \( \preceq_T \) on its nodes, where the smallest element is the root and leaves form maximal elements. For a vertex \( v \in V(G) \), we denote by \( F_v(T) \) the unique \( \preceq_T \)-minimal node \( t \) of \( T \) with \( v \in \chi(t) \), which is well-defined because of Properties (T1) and (T3) of a tree decomposition. Let \( \prec_T \) be the partial ordering of \( V(G) \) such that \( u \prec_T v \) if and only if \( F_u(T) \prec_T F_v(T) \) for every \( u,v \in V(G) \). We say that \((T,\chi)\) is a dependency tree decomposition if it satisfies the following additional property:

(T4) \( \prec_T \) is compatible with \( \leq_P \), i.e., for every two vertices \( u \) and \( v \) of \( G \) it holds that whenever \( F_u(T) \prec_T F_v(T) \) then it does not hold that \( v \leq_P u \).

Lemma 14. A graph \( G \) has a \( P \)-elimination ordering of width at most \( \omega \) if and only if \( G \) has a dependency tree decomposition of width at most \( \omega \). Moreover, a \( P \)-elimination ordering of width \( \omega \) can be obtained from a dependency tree decomposition of width \( \omega \) in polynomial-time and vice versa.

Proof. For the forward direction we will employ the construction given by Kloks in [18], which shows that a normal elimination ordering can be transformed into a tree decomposition of the same width. We will then show that this construction also retains the compatibility with \( P \). Let \( \leq^\varnothing = (v_1, \ldots, v_n) \) be a dependency elimination ordering for \( G \) of width \( \omega \) and let \( H \) be the fill-in graph of \( G \) w.r.t. \( \leq^\varnothing \). We will iteratively construct a sequence \( (T_0, \ldots, T_{n-1}) \) such that for every \( i \) with \( 0 \leq i < n \), \( T_i = (I_i, \chi_i) \) is dependency tree decompositions of the graph \( H_i = H[\{v_{n-i}, \ldots, v_n\}] \) of width at most \( \omega \). Because \( T_{n-1} \) is a dependency tree decomposition of \( H_{n-1} = H \) of width at most \( \omega \), this shows the forward direction of the lemma. In the beginning we set \( T_0 \) to be the trivial tree decomposition of \( H_0 \), which contains merely one node whose bag consists of the vertex \( v_n \). Moreover, for every \( i \) with \( 0 < i < n \), \( T_i \) is obtained from \( T_{i-1} \) as follows. Note that because \( N_{H_i}(v_{n-i}) \) induces a clique in \( H_{i-1} \), \( T_{i-1} \) contains a node that covers all vertices in \( N_{H_i}(v_{n-i}) \). Let \( t \) be any such bag, then is \( T_i \) is obtained from \( T_{i-1} \) by adding a new node \( t' \) to \( T_{i-1} \) making it adjacent to \( t \) and setting \( \chi_i(t') = N_{H_i}[v_{n-i}] \). It is known [18] that \( T_i \) satisfies the Properties (T1)–(T3) of a tree decomposition and it hence only remains to show that \( T_i \) satisfies (T4). Since, by induction hypothesis, \( T_{i-1} \) is a dependency tree decomposition, Property (T4) already holds for every pair \( u,v \in V(H_{i-1}) \). Hence it only remains to consider pairs \( u,v \) for some \( u \in V(H_{i-1}) \). Because the only node containing \( v_{n-i} \) in \( T_i \) is a leaf, we can assume that \( F_u(T) \prec_T F_{v_{n-i}}(T) \) and because \( v_{n-i} \leq^\varnothing u \) it cannot hold that \( v_{n-i} \leq_P u \), as required.

For the reverse direction, let \( T = (T,\chi) \) be a \( P \)-tree decomposition of \( G \) of width at most \( \omega \). It is known [18] that any linear extension of \( \prec_T \) is an elimination ordering for \( G \) of width at most \( \omega \). Moreover, because of Property (T4), \( \prec_T \) is compatible with \( \leq_P \) and hence there is a linear extension of \( \prec_T \), which is also a linear extension of the reverse of \( \leq_P \).

Dependency cops and robber game: Recalling the definition of the (monotone) cops and robber game for treewidth, we define the dependency cops and robber game (for a QBF instance \( I \) with dependency poset \( P \)) analogously but with the additional restriction that legal moves must also satisfy a third condition:

CM3 \( C' \setminus C \) is downward-closed in \( R \), i.e., there is no \( r \in R \setminus C' \) with \( r \leq_P c \) for any \( c \in C \setminus C' \). Intuitively, condition CM3 restricts the cop-player by forcing him to search vertices (variables) in an order that is compatible with the dependency poset.

To formally prove the equivalence between the cop-number for this restricted game and dependency treewidth, we will need to also formalize the notion of a strategy. Here we will represent strategies for the cop-player as rooted trees whose nodes are labeled with
positions for the cop-player and whose edges are labeled with positions for the robber-
player. Namely, we will represent winning strategies for the cop-player on a primal graph
G by a triple \((T, \alpha, \beta)\), where \(T\) is a rooted tree, \(\alpha : V(T) \to 2^{V(G)}\) is a mapping from the
nodes of \(T\) to subsets of \(V(G)\), and \(\beta : E(T) \to 2^{V(G)}\), satisfying the following conditions:

\(\text{CS1} \) \(\alpha(r) = \emptyset\) and for every component \(R\) of \(G\), the root node \(r\) of \(T\) has a unique child \(c\)
with \(\beta([r, c]) = R\), and

\(\text{CS2} \) for every other node \(t\) of \(T\) with parent \(p\) it holds that: the move from position
\((\alpha(p), \beta([p, t]))\) to position \((\alpha(t), \beta([t, c]))\) is legal for every child \(c\) of \(t\) and moreover
for every component \(R\) of \(G \setminus \alpha(t)\) contained in \(\beta([p, t])\), \(t\) has a unique child \(c\) with
\(\beta([t, c]) = R\).

Informally, the above properties ensure that every play consistent with the strategy is win-
ning for the cop-player and moreover for every counter-move of the robber-player, the
strategy gives a move for the cop-player. The width of a winning strategy for the cop-
player is the maximum number of cops simultaneously placed on \(G\) by the cop-player, i.e.,
\(\max_{t \in V(T)} |\alpha(t)|\). The cop-number of a graph \(G\) is the minimum width of any winning strat-
 egy for the cop-player on \(G\). We are now ready to show the equivalence between dependency
tree decompositions and winning strategies for the cop-player.

\(\blacktriangleright \text{Lemma 15.} \) For every graph \(G\) the width of an optimal dependency tree decomposition
plus one is equal to the cop-number of the graph. Moreover, a dependency tree decomposition
of width \(\omega\) can be obtained from a winning strategy for the cop-player of width \(\omega + 1\) in
polynomial-time and vice versa.

\(\text{Proof.}\) Let \(\mathcal{T} = (T, \chi)\) be a dependency tree decomposition of \(G\) of width \(\omega\). We start
by showing that \(\mathcal{T}\) can be transformed into a dependency tree decomposition of width \(\omega\)
satisfying:

\((*)\) \(\chi(r) = \emptyset\) for the root node \(r\) of \(T\) and for every node \(t \in V(T)\) with child \(c \in V(T)\) in
\(T\) the set \(\chi(T_c) \setminus \chi(t)\) is a component of \(G \setminus \chi(t)\).

To ensure that \(\mathcal{T}\) satisfies \((*)\) it is sufficient to add a new root vertex \(r'\) to \(T\) and
set \(\chi(r') = \emptyset\). We show next that starting from the root of \(T\) we can ensure that for every
node \(t \in V(T)\) with child \(c\) the set \(\chi(T_c) \setminus \chi(t)\) is a component of \(G \setminus \chi(t)\). Let \(t\) be a
node with child \(c\) in \(T\) for which this does not hold. Because of Proposition 4, we have
that \(\chi(T_c) \setminus \chi(t)\) is a set of components, say containing \(C_1, \ldots, C_l\) of \(G \setminus \chi(t)\). For every
\(i\) with \(1 \leq i \leq l\), let \(T_i = (T_i, \chi_i)\) be the dependency tree decomposition with \(T_i = T_c\) and
\(\chi_i(t') = \chi(t') \cap (C_i \cup \chi(t))\) and root \(r_i = c\). Then we replace the entire sub dependency
tree decomposition of \(\mathcal{T}\) induced by \(T_c\) in \(T\) with the tree decompositions \(T_1, \ldots, T_l\) such that
\(t\) now becomes adjacent to the roots \(r_1, \ldots, r_l\). It is straightforward to show that the
result of this operation is again a dependency tree decomposition of \(G\) of width at most \(\omega\)
and moreover the node \(t\) has one child less that violates \((*)\). By iteratively applying this
operation to every node \(t\) of \(\mathcal{T}\) we eventually obtain a dependency tree decomposition that
satisfies \((*)\).

Hence w.l.o.g. we can assume that \(\mathcal{T}\) satisfies \((*)\). We now claim that \((T, \alpha, \beta)\) where:

\(- \alpha(t) = \chi(t)\) for every \(t \in V(T)\),
\(- \alpha(t) = \chi(t)\) for every \(t \in V(T)\) with parent \(p \in V(T)\), \(\beta([p, t]) = \chi(T_p) \setminus \chi(p)\).

is a winning strategy for \(\omega + 1\) cops. Observe that because \(\mathcal{T}\) satisfies \((*)\), it holds that
\(\alpha(r) = \emptyset\) and for every \(t \in V(T)\) with parent \(p \in V(T)\), the pair \((\alpha(p), \beta([p, t]))\) is a position
in the visible \(P\)-cops and robber game on \(G\).

We show next that for every \(t, p\) as above and every child \(c\) of \(t\) in \(T\), it holds that the move
from \((\alpha(p), \beta([p, t]))\) to \((\alpha(t), \beta([t, c]))\) is valid. Because \(\beta([t, c]) = \chi(T_c) \setminus \chi(t) \subseteq \chi(T_i) \setminus \chi(t)\)
\( \chi(p) = \beta(\{p, t\}) \) and \( \beta(\{p, t\}) \) is connected in \( G \setminus \alpha(p) \) (and hence also in \( G \setminus (\alpha(p) \cap \alpha(t)) \)), it follows that \( \beta(\{p, t\}) \) and \( \beta(\{t, c\}) \) are contained in the same component of \( G \setminus (\alpha(p) \cap \alpha(t)) \), which shows CM1. Because of Proposition \( \text{[1]} \) it holds that \( \alpha(p) \cap \alpha(t) \) and hence in particular \( \alpha(t) \) separates \( \beta(\{p, t\}) \) from the rest of the graph, which shows CM2, i.e., \( \delta(\beta(\{p, t\})) \subseteq \alpha(t) \).

Towards showing CM3 suppose for a contradiction that there is a \( r \in \beta(\{p, t\}) \setminus \alpha(t) \) with \( r \leq \{c \) for some \( c \in \alpha(t) \setminus \alpha(p) \). Because \( r \in \beta(\{p, t\}) \setminus \alpha(t) \) and \( c \in \alpha(t) \), we obtain that \( c \prec_r \{c \) and because of (T4) it follows that \( r \leq \{c \) does not hold. Note that CS1 and the second part of CS2, i.e., for every node \( t \) with parent \( p \) and every component \( R \in G \setminus \alpha(t) \), \( t \) has a unique child \( c \) with \( \beta(\{t, c\}) = R \), both hold because \( T \) satisfies T1 and T3 (T3 is only needed to show that the child is unique).

On the other hand, let \( S = (T, \alpha, \beta) \) be a winning strategy for the cop-player in the visible \( P \)-cops and robber game on \( G \) using \( \omega \) cops. Observe that \( S \) can be transformed into a winning strategy for the cop-player using \( \omega \) cops satisfying:

\( \text{(a)} \) for every node \( t \) of \( T \) with parent \( p \) it holds that \( \alpha(t) \subseteq \delta(\beta(\{p, t\})) \cup \beta(\{p, t\}) \).

Indeed; if \( \text{(a)} \) is violated, then one can simply change \( \alpha(t) \) to \( \alpha(t) \cap (\delta(\beta(\{p, t\})) \cup \beta(\{p, t\})) \) without violating any of CS1 or CS2. Hence we can assume that \( S \) satisfies \( \text{(a)} \).

We now claim that \( T = (T, \alpha) \) is a dependency tree decomposition of \( G \) of width \( \omega - 1 \). Towards showing T1, let \( v \in V(G) \). Because of CS1, it holds that either \( v \in \alpha(r) \) for the root \( r \) of \( T \) or there is a child \( c \) of \( r \) in \( T \) with \( v \in \beta(\{r, c\}) \). Moreover, due to CS2 we have that either \( v \in \alpha(c) \) or \( v \in \beta(\{c, c'\}) \) for some child \( c' \) of \( c \) in \( T \). By proceeding along \( T \), we will eventually find a node \( t \in V(T) \) with \( v \in \alpha(t) \). Towards showing T2, let \( \{u, v\} \in E(G) \).

Again because of CS1, it holds that either \( \{u, v\} \subseteq \alpha(r) \), or \( \{u, v\} \subseteq \delta(\beta(\{r, c\})) \cup \beta(\{r, c\}) \) for some child \( c \) of \( r \) in \( T \). Because of CM2, we obtain that \( \delta(\beta(\{r, c\})) \subseteq \alpha(c) \) and together with CS2, we have that either \( \{u, v\} \subseteq \alpha(c) \) or \( \{u, v\} \subseteq \delta(\beta(\{c, c'\})) \cup \beta(\{c, c'\}) \) for some child \( c' \) of \( c \) in \( T \). By proceeding along \( T \), we will eventually find a node \( t \in V(T) \) with \( \{u, v\} \subseteq \alpha(t) \).

Next, we prove that T3 holds by showing that for any three nodes \( t_1, t_2, \) and \( t_3 \) in \( T \) such that \( t_2 \) lies on the unique path from \( t_1 \) to \( t_3 \) in \( T \), it holds that \( \alpha(t_1) \cap \alpha(t_3) \subseteq \alpha(t_2) \). We will distinguish two cases: (1) \( t_1 \) is not an ancestor of \( t_3 \) and vice versa, and (2) \( t_1 \) and \( t_2 \) lie on the unique path from the root of \( T \) to \( t_3 \).

Before proceeding, we need to establish that \( S \) satisfies the following property:

\( \text{(b)} \) for every node \( t \) with child \( c \) in \( T \) it holds that \( \bigcup_{t' \in V(T_c)} \alpha(t') \subseteq \delta(\beta(\{t, c\})) \cup \beta(\{t, c\}) \).

Because of CM2 we have that \( \beta(\{t, c\}) \subseteq \beta(\{p, t\}) \) for every three nodes \( p, t, \) and \( c \) such that \( p \) is the parent of \( t \) which in turn is the parent of \( c \) in \( T \). Moreover, because of \( \text{(a)} \) we have that \( \alpha(t) \subseteq \delta(\beta(\{p, t\})) \cup \beta(\{p, t\}) \) for every node \( t \) with parent \( p \) in \( T \). Applying these two facts iteratively along a path from \( t \) to any of its descendants \( t' \) in \( T \), we obtain that \( \alpha(t) \subseteq \delta(\beta(\{p, t\})) \cup \beta(\{p, t\}) \), as required.

Towards showing case (1) assume that this is not the case, i.e., there are \( t_1, t_2, \) and \( t_3 \) as above and a vertex \( v \in \alpha(t_1) \cap \alpha(t_3) \) but \( v \notin \alpha(t_2) \). W.l.o.g. we can assume that \( t_2 \) is the least common ancestor of \( t_1 \) and \( t_2 \) in \( T \), otherwise we end up in case (2). Let \( c_1 \) and \( c_2 \) be the two children of \( t_2 \) such that \( t_1 \in V(T_{c_1}) \) and \( t_2 \in V(T_{c_2}) \). Then because \( v \in \alpha(t_1) \) we obtain from \( \text{(b)} \) that \( v \in \delta(\beta(\{t_2, c_1\})) \cup \beta(\{t_2, c_1\}) \) for any \( i \in \{1, 2\} \).

Because the sets \( \beta(\{t_2, c_1\}) \) and \( \beta(\{t_2, c_2\}) \) are disjoint, it follows that \( v \in \delta(\beta(\{t_2, c_1\})) \) and thus \( v \in \alpha(t_2) \), contradicting our assumption that this is not the case.

Towards showing case (2) assume that this is not the case, i.e., there are \( t_1, t_2, \) and \( t_3 \) as above and a vertex \( v \in \alpha(t_1) \cap \alpha(t_3) \) but \( v \notin \alpha(t_2) \). W.l.o.g. we can assume that \( t_2 \) is a child of \( t_1 \) in \( T \). Because \( v \in \alpha(t_3) \) we obtain from \( \text{(b)} \) that \( \alpha(t_3) \subseteq \delta(\beta(\{t_1, t_2\}) \cup \beta(\{t_1, t_2\}) \).

Hence either \( v \in \delta(\beta(\{t_1, t_2\})) \) or \( v \in \beta(\{t_1, t_2\}) \). In the former case it follows from CM2 that \( v \in \alpha(t_2) \) a contradiction to our assumption that \( v \notin \alpha(t_2) \) and in the latter case, we obtain that \( v \notin \alpha(t_1) \) a contradiction to our assumption that \( v \in \alpha(t_1) \).
Towards showing \( T4 \) assume that this is not the case, i.e., there are \( u, v \in V(G) \) with \( u <_T v \) but \( v \not\leq^P u \). Let \( p \) be the parent of \( F_u(T) \) in \( T \). Then because \( u, v \in \bigcup_{t' \in V(T_F_u(T))} \alpha(t') \) and (b), we obtain that \( u, v \in \delta(\beta(\{p, F_u(T)\})) \cup \beta(\{p, F_u(T)\}) \). Moreover, since neither \( u \) or \( v \) are in \( \alpha(p) \), we have that \( u, v \in \beta(\{p, F_u(T)\}) \). Finally because \( u \in \alpha(F_u(T)) \) but not \( v \in \alpha(F_u(T)) \), we obtain from CM3 that \( u \not\leq^P v \) contradicting our assumption that \( v \not\leq^P u \).

5 Computing Dependency Treewidth

In this section we will present two exact algorithms to compute dependency treewidth. The first algorithm is based on the characterization of dependency treewidth in terms of the cops and robber game and shows that, for every fixed \( \omega \), determining whether a graph has dependency treewidth at most \( \omega \), and in the positive case also computing a dependency tree decomposition of width at most \( \omega \), can be achieved in polynomial time. The second algorithm is based on a chain partition of the given dependency poset and shows that if the width of the poset is constant, then an optimal dependency tree decomposition can be constructed in polynomial time.

Before proceeding to the algorithms, we would like to mention here that the fixed-parameter algorithm for computing first order treewidth \([1]\) can also be used for computing dependency treewidth in the restricted case that the trivial dependency poset is used.

Below we provide our first algorithm for the computation of dependency treewidth.

\textbf{Theorem 16.} There is an algorithm running in time \( \mathcal{O}(|V(G)|^{2\omega+2}) \) that, given a graph \( G \) and a poset \( \mathcal{P} = (V(G), \leq^P) \) and \( \omega \in \mathbb{N} \), determines whether \( \omega \) cops have a winning strategy in the dependency cops and robber game on \( G \) and \( \mathcal{P} \), and if so outputs such a winning strategy.

\textbf{Proof.} The idea is to transform the cops and robber game on \( G \) into a much simpler two player game, the so-called simple two player game, which is played on all possible positions of the cops and robber game on \( G \).

A \textit{simple two player game} is played between two players, which in association to the cops and robber game, we will just call the cops and the robber player \([15]\). Both players play by moving a token around on a so-called arena, which is a triple \( A = (V_C, V_R, A) \) such that \( (V_C \cup V_R, A) \) is a bipartite directed graph with bipartition \( (V_C, V_R) \). The vertices in \( V_C \) are said to belong to the cop-player and the vertices in \( V_R \) are said to belong to the robber-player. Initially, one token is placed on a distinguished starting vertex \( s \in V_C \cup V_R \).

From then onward the player who owns the vertex, say \( v \), that currently contains the token, has to move the token to an arbitrary successor (i.e., out-neighbor) of \( v \) in \( A \). The cop-player wins if the robber-player gets stuck, i.e., the token ends up in a vertex owned by the robber-player that has no successors in \( A \), otherwise the robber-player wins. It is well-known that strategies in this game are deterministic and memoryless, i.e., strategies for a player are simple functions that assign every node owned by the player one of its successors. Moreover, the winning region for both players as well as their corresponding winning strategy can be computed in time \( \mathcal{O}(|V_C \cup V_R| + |A|) \) by the following algorithm. The algorithm first computes the winning region \( W_C \), as follows.

Initially all vertices owned by the robber-player which do not have any successors in \( A \) are placed in \( W_C \). The algorithm then iteratively adds the following vertices to \( W_C \):

- all vertices owned by the cop-player that have at least one successor \( W_C \),
- all vertices owned by the robber-player for which all successors are in \( W_C \).
Once the above process stops, the set $W_C$ is the winning region of the cop-player in $A$ and $(V_C \cup V_R) \setminus W_C$ is the winning region for the robber-player. Moreover, the winning strategy for both players can now be obtained by choosing for every vertex a successor that is in the winning region of the player owning that vertex (if no such vertex exists, then an arbitrary successor must be chosen).

Given a graph $G$, a poset $\mathcal{P} = (V(G), \leq)$, and an integer $\omega$, we construct an arena $A = (V_C, V_R, A)$ and a starting vertex $s \in V_R$ such that $\omega$ cops have a winning strategy in the $\mathcal{P}$-cops and robber game on $G$ iff the cop-player wins from $s$ in the simple two player game on $A$ as follows:

- We set $V_C$ to be the set of all pairs $(C, R)$ such that $(C, R)$ is a position in the $\mathcal{P}$-cops and robber game on $G$ using at most $\omega$ cops ($|C| \leq \omega$).
- We set $V_R$ to be the set of all triples $(C, C', R)$ such that:
  - $(C, R)$ is a position in the $\mathcal{P}$-cops and robber game on $G$ using at most $\omega$ cops ($|C| \leq \omega$), and
  - $C' \subseteq V(G)$ is a potential new cop-position for at most $\omega$ cops from $(C', R')$, i.e., $\delta(R) \subseteq C'$ and $C, R, C'$ satisfy CM3.
- From every vertex $(C, R) \in V_C$ we add an arc to all vertices $(C, C', R) \in V_R$.
- From every vertex $(C, C', R) \in V_R$ we add an arc to all vertices $(C', R') \in V_C$ such that the move from $(C, R)$ to $(C', R')$ is legal.
- Additionally $V_R$ contains the starting vertex $s$ that has an outgoing arc to every vertex $(\emptyset, R) \in V_C$ such that $R$ is a component of $G$.

By construction $|V_C| \leq |V(G)|^{2\omega+1}$ and $|V_R| \leq |V(G)|^{2\omega+1}$. Moreover, because every vertex in $V_C$ has at most $|V(G)|^\omega$ successors and every vertex in $V_R$ has at most $|V(G)|^\omega$ successors, we obtain that $|A| \leq |V(G)|^{2\omega+2}$. Let us now analyze the running time required to construct $A$. We can construct all vertices $(C, C', R) \in V_C$ in time $O(|V(G)|^\omega|E(G)|)$ by computing the set of all components of $G \setminus C$ for every cop-position $C$. Note that within the same time we can additionally compute and store the guards $\delta(R)$ for every component $R$. For each vertex in $(C, R) \in V_C$ we can then compute the associated vertices $(C, C', R) \in V_R$ and add the necessary arcs by enumerating all sets $C' \subseteq V(G)$ with $\delta(R) \subseteq C'$ and checking for each of those whether $R$ and $C'$ satisfy CM3. Enumerating the sets $C' \subseteq V(G)$ with $\delta(R) \subseteq C'$ can be achieved in time $O(|V(G)|^\omega)$ (using the fact that we stored $\delta(R)$ for every component $R$). Moreover, determining whether $R$ and $C'$ satisfy CM3 can be achieved in time $O(\omega|R|) = O(\omega|V(G)|)$ by going over all vertices $r \in R$ and verifying that $r \in C'$ or that it is not smaller than any vertex in $C'$. Hence computing all vertices of $A$ and all arcs from vertices in $V_C$ to vertices in $V_R$ can be achieved in time at most $O(|V(G)|^\omega|E(G)| + |V(G)|^{2\omega+1}\omega)$, which for the natural assumption that $\omega > 0$ is at most $O(|V(G)|^{2\omega+2})$. Finally, we need to add the arcs between vertices in $(C, C', R) \in V_R$ and the vertices $(C', R') \in V_C$. Note that there is an arc from $(C, C', R) \in V_R$ to $(C', R') \in V_C$ if and only if $R' \subseteq R$ and moreover for every component $R'$ of $G \setminus C'$ either it is a subset of $R$ or it is disjoint with $R$, which can be checked in constant time. Hence the total time required for this last step is equal to the number of vertices in $V_R$ times $|V(G)|$ which is at most $O(|V(G)|^{2\omega+2})$. It follows that the time required to construct $A$ is at most $O(|V(G)|^{2\omega+2})$. Once the arena is constructed the winning regions as well as the winning strategies for both players can be computed in time $O(|V_C \cup V_R| + |A|) = O(|V(G)|^{2\omega+2})$.

The next theorem summarizes our second algorithm for computing dependency treewidth. The core distinction here lies in the fact that the running time does not depend on the dependency treewidth, but rather on the poset-width. This means that the algorithm can
To decide whether \( G \) has a dependency elimination ordering of width at most \( \omega \), we first build an auxiliary directed graph \( H \) as follows.

The vertex set of \( H \) consists of all pairs \( (D, d) \) such \( D \subseteq V(G) \) is a downward closed set and \( d \in D \) is a maximal element of \( D \) such that \( |N_G[D \setminus d]| \leq \omega \). Additionally, \( H \) contains the vertices \((V(G), \emptyset)\) and \((\emptyset, \emptyset)\). Furthermore, there is an arc from \((D, d)\) to \((D', d')\) of \( H \) if and only if \( D' = D \cup \{d'\} \) or \( D = D' = V(G) \) and \( d' = \emptyset \). This completes the construction of \( H \). It is immediate that \( G \) has a dependency elimination ordering of width at most \( \omega \) if and only if there is a directed path in \( H \) from \((\emptyset, \emptyset)\) to \((V(G), \emptyset)\). Hence, given \( H \) we can decide whether \( G \) has a dependency elimination ordering of width at most \( \omega \) (and output it, if it exists) in time \( O(|V(H)| \log(|V(H)|) + E(H)) \) (e.g., by using Dijkstra’s algorithm).

Let \( k \) be the width of the poset \( P \). Due to Proposition 1 we can compute a chain partition \( C = (W_1, \ldots, W_k) \) of width \( k \) in time \( O(k \cdot |V(G)|^2) \). Note that every downward closed \( D \) set can be characterized by the position of the maximal element in \( D \) on each of the chains \( W_1, \ldots, W_k \), we obtain that there are at most \(|V(G)|^k\) downward closed sets. Hence, \( H \) has at most \( O(|V(G)|^k(k + 1)) \) vertices its vertex set can be constructed in time \( O(|V(G)|^k(k + 1)) \). Since every vertex \((D, d)\) of \( H \) has at most \( k + 1 \) possible out-neighbors, we can construct the arc set of \( H \) in time \( O(|V(G)|^k k^2) \).

Hence, the total time required to construct \( H \) is \( O(|V(G)|^k k^2) \) which dominates the time required to find a shortest path in \( H \), and so the runtime follows. ▶

6 Concluding Notes

Dependency treewidth is a promising decompositional parameter for QBF which overcomes the key shortcomings of previously introduced structural parameters; its advantages include a single-exponential running time, a refined and flexible approach to variable dependencies, and the ability to compute decompositions. It also admits several natural characterizations that show the robustness of the parameter and allows the computation of resolution proofs.

The presented algorithms for computing dependency elimination orderings leave open the question of whether this problem admits a fixed-parameter algorithm (parameterized by dependency treewidth). We note that the two standard approaches for computing treewidth fail here. In particular, the well-quasi-ordering approach with respect to minors does not work since the set of ordered graphs can be observed not to be well-quasi ordered w.r.t. the ordered minor relation \([29]\). On the other hand, the records used in the second approach \([3]\) do not provide sufficient information in our ordered setting.

6.0.0.1 Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments. Eduard Eiben acknowledges support by the Austrian Science Fund (FWF, projects P26096 and W1255-N23). Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.
References

1. Isolde Adler and Mark Weyer. Tree-width for first order formulae. *Logical Methods in Computer Science*, 8(1), 2012.
2. Albert Atserias and Sergi Oliva. Bounded-width QBF is pspace-complete. *J. Comput. Syst. Sci.*, 80(7):1415–1429, 2014.
3. Armin Biere and Florian Lonsing. Integrating dependency schemes in search-based QBF solvers. In *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *LNCS*, pages 158–171. Springer, 2010.
4. Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, May 16-18, 1993, San Diego, CA, USA, pages 226–234, 1993.
5. Florent Capelli. Structural restrictions of CNF-formulas: applications to model counting and knowledge compilation. PhD thesis, Université Paris Diderot, 2016.
6. Hubie Chen and Victor Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. *SIAM J. Comput.*, 45(6):2066–2086, 2016.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. of the ACM*, 7(3):201–215, 1960.
8. Reinhard Diestel. Graph minors 1: A short proof of the path-width theorem. *Combinatorics, Probability & Computing*, 4:27–30, 1995.
9. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
10. Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.
11. Uwe Egly, Thomas Eiter, Hans Tompits, and Stefan Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, July 30 - August 3, 2000, Austin, Texas, USA., pages 417–422, 2000.
12. Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. Using decomposition-parameters for QBF: mind the prefix! In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, February 12-17, 2016, Phoenix, Arizona, USA., pages 964–970, 2016.
13. S. Felsner, V. Raghavan, and J. Spinrad. Recognition algorithms for orders of small width and graphs of small Dilworth number. *Order*, 20(4):351–364, 2003.
14. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.
15. Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
16. Hans Kleine Büning and Uwe Bubeck. Theory of quantified boolean formulas. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 23, pages 735–760. IOS Press, 2009.
17. Hans Kleine Büning and Theodor Lettman. *Propositional logic: deduction and algorithms*. Cambridge University Press, Cambridge, 1999.
18. T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, Berlin, 1994.
19. Charles Otwell, Anja Remshagen, and Klaus Truemper. An effective QBF solver for planning problems. In *Proceedings of the International Conference on Modeling, Simulation & Visualization Methods, MSV '04 & Proceedings of the International Conference on Algorithmic Mathematics & Computer Science, AMCS '04, June 21-24, 2004, Las Vegas, Nevada, USA*, pages 311–316. CSREA Press, 2004.
20. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
21 Luca Pulina and Armando Tacchella. A structural approach to reasoning with quantified boolean formulas. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 596–602, 2009.

22 Luca Pulina and Armando Tacchella. An empirical study of QBF encodings: from treewidth estimation to useful preprocessing. *Fundam. Inform.*, 102(3-4):391–427, 2010.

23 J. Rintanen. Constructing conditional plans by a theorem-prover. *J. Artif. Intell. Res.*, 10:323–352, 1999.

24 Ashish Sabharwal, Carlos Ansótegui, Carla P. Gomes, Justin W. Hart, and Bart Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Theory and Applications of Satisfiability Testing - SAT 2006*, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, *Proceedings*, pages 382–395, 2006.

25 Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Autom. Reasoning*, 42(1):77–97, 2009.

26 Friedrich Slivovsky and Stefan Szeider. Computing resolution-path dependencies in linear time. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 58–71. Springer Verlag, 2012.

27 Friedrich Slivovsky and Stefan Szeider. Variable dependencies and q-resolution. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2014.

28 Friedrich Slivovsky and Stefan Szeider. Quantifier reordering for QBF. *J. Autom. Reasoning*, 56(4):459–477, 2016.

29 Daniel A Spielman and Miklós Bóna. An infinite antichain of permutations. *Electron. J. Combin.*, 7:N2, 2000.

30 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973.

31 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer Verlag, 2004.

32 Allen Van Gelder. Variable independence and resolution paths for quantified boolean formulas. In Jimmy Lee, editor, *Principles and Practice of Constraint Programming - CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 789–803. Springer Verlag, 2011.