Small Resolution Proofs for QBF using Dependency Treewidth

Eduard Eiben
Algorithms and Complexity Group, TU Wien, Vienna, Austria and
Department of Informatics, University of Bergen, Norway
eduard.eiben@uib.no

Robert Ganian
Algorithms and Complexity Group, TU Wien, Vienna, Austria and
FI MU, Brno, Czech Republic
rganian@gmail.com

Sebastian Ordyniak
Algorithms and Complexity Group, TU Wien, Vienna, Austria
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.

2012 ACM Subject Classification Theory of computation → Logic, Theory of computation → Fixed parameter tractability

Keywords and phrases QBF, Treewidth, Fixed Parameter Tractability, Dependency Schemes

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.28

Related Version The full version of this paper is available at `https://arxiv.org/abs/1711.02120`, [13].

Funding Eduard Eiben was supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744) and by the Austrian Science Fund (FWF, projects P26696 and W1255-N23).

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, 20, 24, 25]. QBF is the archetypical PSPACE-complete problem and is therefore believed to be computationally harder than NP-complete problems such as SAT [18, 21, 31].

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, 32] 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 $f(k) \cdot n$ [32] for a computable function f. Algorithms with running time in this form (i.e., $f(k) \cdot n^{\mathcal{O}(1)}$, 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* [8, 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 [1, 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 [22, 23].

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 allows a $\mathcal{O}(3^{2k}nk)$-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 [26, 29], 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* [26], 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. 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.

2 Preliminaries

For $i \in \mathbb{N}$, we let $[i]$ denote the set $\{1, \dots, i\}$. We refer to the book by Diestel [9] 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, 15] for an in-depth overview of parameterized complexity theory. Here, we only recall that a *parameterized problem* (Q, κ) is a *problem* $Q \subseteq \Sigma^*$ together with a *parameterization* $\kappa \colon \Sigma^* \to \mathbb{N}$, where Σ is a finite alphabet. A parameterized problem (Q, κ) is *fixed-parameter tractable (w.r.t. κ)*, in short *FPT*, if there exists a decision algorithm for Q, a computable function f, and a polynomial function p, 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 ϕ, we denote the set of variables which occur in ϕ 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 (ϕ, τ) where ϕ is a CNF formula and τ is a sequence of quantified variables, denoted $\text{var}(\tau)$, which satisfies $\text{var}(\tau) \supseteq \text{var}(\phi)$; then ϕ is called the *matrix* and τ is called the *prefix*. A QBF (ϕ, τ) 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 ϕ, and st is an edge in G_I if there exists a clause in ϕ 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 $(W_1, \dots, W_k)$ such that $\{W_1, \dots, W_k\}$ 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 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 $\mathcal{P} = (P, \leq^P)$ is a relation $\preceq$ over P such that $x \preceq y$ whenever $x \leq^P y$ and the poset $\mathcal{P}^* = (P, \preceq)$ 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 $\mathcal{V} := (V, \leq^V)$.

We use *dependency posets* to provide a general and formal way of speaking about the various *dependency schemes* introduced for QBF [26]. 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* [28], 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* $\mathcal{V} = (\text{var}(\phi), \leq^I)$ of I is a poset over var(ϕ) 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 $\mathcal{V}$, the QBF $I' = (\phi, \tau_{\preceq})$, obtained by permutation of the prefix τ 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 [26, 28]. In particular, it is easy to verify that a dependency poset can be computed from any permutation dependency scheme in polynomial time.

To illustrate these definitions, consider the following QBF: $\exists a \forall b \exists c (a \vee c) \wedge (b \vee c)$. 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 [33, 27]) 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 [17]. 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 ϕ_x be the set of all clauses in ϕ 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 ϕ' from ϕ by removing all occurrences of x and $\bar{x}$.

▶ **Lemma 1.** *Let $I = (\phi, \tau)$ and $x \in var(\phi)$ be the last variable in τ. If x is existentially quantified, then I is true if and only if $res(I, x)$ is true.*

▶ **Lemma 2.** *Let $I = (\phi, \tau)$ and $x \in var(\phi)$ be the last variable in τ. If x is universally quantified, then I is true if and only if $I \setminus x$ is true.*

2.4 Treewidth

Here we will introduce three standard characterizations of treewidth [19]: tree decompositions, elimination orderings, and cops and robber games. These will play a 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, χ), where T is a rooted tree and χ 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 $\mathcal{T}$ 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.

Elimination ordering. An *elimination ordering* of a graph is a linear order of its vertices. Given an elimination ordering ϕ of the graph G, the *fill-in graph* H of G w.r.t. ϕ is the unique minimal graph such that: $V(G) = V(H)$, $E(H) \supseteq E(G)$, and if $0 \leq k < i < j \leq n$ and $v_i, v_j \in N_H(v_k)$, then $v_iv_j \in E(H)$. The *width* of elimination ordering ϕ is the maximum number of neighbors of any vertex v that are larger than v (w.r.t. ϕ) 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:

CM1 R and R' are contained in the same component of $G \setminus (C \cap C')$,
CM2 $\delta(R) \subseteq C'$.

A play $\mathcal{P}$ is a sequence $(\emptyset, R_0), \ldots, (C_n, R_n)$ 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 $\max_{i \leq n} |C_i|$. A play $\mathcal{P}$ 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.

For any graph G it holds that G has treewidth k iff G has an elimination ordering of width k iff G has cop-number k [19].

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. Furthermore, we remark that on existentially quantified QBFs dependency treewidth w.r.t. trivial dependency poset collapses with classical 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.

▶ **Theorem 3.** *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 π of width k, decides whether I is true in time $\mathcal{O}(3^{2k}kn)$. Moreover, if I is false, then the algorithm outputs a Q-resolution refutation of size $\mathcal{O}(3^k n)$.*

Sketch of Proof. Let $I = (\phi, \tau)$ and let $x_1, \dots, x_n$ denote the variables of ϕ 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 π is a dependency elimination ordering, it follows that the QBF instance $I' = (\phi, \tau')$, where τ' is the reverse of π, is true if and only if I is true.

To solve I we use a modification of the Davis Putnam resolution algorithm [8]. We start with instance I' and recursively eliminate the last variable in the prefix using Lemmas 1 and 2 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 π, 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) $I_{i+1} = res(I_i, x_i)$, 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 1 and 2 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. One can 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 ϕ^i_x and $\phi^i_{\bar{x}}$ contain at most 3^k clauses, and this is sufficient to ensure a small Q-resolution refutation if the instance is false. A formal proof of this fact and runtime analysis are provided in the full version. ◀

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. We remark that we do not include formal definitions of the parameters in this section, since they are technical and not critical for our exposition.

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 4 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 3), while the latter uses a prohibitive triple-exponential algorithm [12].

▶ **Lemma 4.** *Let us fix the trivial dependency poset. There exist infinite classes $\mathcal{A}, \mathcal{B}$ of QBF instances such that:*

a. *$\mathcal{A}$ has unbounded dependency treewidth but prefix pathwidth at most 1;*
b. *$\mathcal{B}$ has unbounded prefix pathwidth (and prefix treewidth) but dependency treewidth at most 1.*

Sketch of Proof. Consider the following examples for $\mathcal{A}, \mathcal{B}$. Let $\mathcal{A} = \{A_i = \exists x_1, \dots, x_i \forall y \exists x\ (y \vee x) \wedge \bigwedge_{j=1}^{i}(x_j \vee x)\}$, and let $\mathcal{B} = \{B_i = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \exists x_{2^i-1} \forall x_{2^i} \exists x_{2^i+1} \bigwedge_{j=1}^{i-1}((x_j \vee x_{2j}) \wedge (x_j \vee x_{2j+1}))\}$. It is straightforward to verify that these classes satisfy the conditions stipulated in the lemma. ◀

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 [33, 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 5.** *There exists an infinite class $\mathcal{C}$ of QBF instances such that $\mathcal{C}$ 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.*

Sketch of Proof. It suffices to set $\mathcal{C}$ to be equal to the class $\mathcal{A}$ used in the proof of the previous lemma and then verify that $\mathcal{C}$ has the desired properties. ◀

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[1]) [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 3. We note that this readily-obtained variant of Theorem 3 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].

▶ **Lemma 6.** *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.*

[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).

Sketch of Proof. It suffices to set $\mathcal{D}$ to be equal to the class $\mathcal{A}$ used in the proof of Lemma 4 and then observe that the same class of instances is used as an example of a class with unbounded CD-width by Chen, Dalmau [6, Example 3.6]. On the other hand, it easy to verify that the QBF instance $E_i = \forall x_1 \forall x_2 \cdots \forall x_i \bigwedge_{1 \leq p < q \leq i}(x_p \vee x_q)$ has CD-width 0, as E_i does not contain an existential variable. However, the primal graph of E_i is a clique and hence it has dependency treewidth $i-1$. ◀

▶ **Lemma 7.** *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.*

Sketch of Proof. Let $F_i = \forall x_{2i} y_{2i} \exists x_{2i-1} y_{2i-1} \cdots \forall x_2 y_2 \exists x_1 y_1 \forall z (z \vee x_1 \vee y_1) \bigwedge_{j=1}^{2i-1}[(x_j \vee x_{j+1}) \wedge (y_j \vee y_{j+1})] \wedge \bigwedge_{j=1}^{i}(x_{2j-1} \vee y_1)$. It is readily observed that the elimination ordering $z x_1 x_2 \ldots x_{2i} y_1 y_2 \ldots y_{2i}$ of width 2 is compatible with the resolution-path dependency poset for this formula. On the other hand, the elimination ordering obtained from first order treewidth is forced to eliminate y_1 before x_j, $j \geq 2$ (see Definitions 3.3, 3.10, 3.15, together with the definitions on page 5 of Adler and Weyer [1]). Therefore, the first order treewidth of this instance would be at least $i-1$. ◀

4 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.

Dependency tree decomposition. Let I be a QBF instance with primal graph G and dependency poset $\mathcal{P}$ and let (T, χ) be a tree decomposition of G. Note that the rooted tree T naturally induces a partial order $\leq_T$ 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 $\leq_T$-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 $<_{\mathcal{T}}$ be the partial ordering of $V(G)$ such that $u <_{\mathcal{T}} v$ if and only if $F_u(T) <_T F_v(T)$ for every $u, v \in V(G)$. We say that (T, χ) is a *dependency tree decomposition* if it satisfies the following additional property:

(T4) $<_{\mathcal{T}}$ is compatible with $\leq^{\mathcal{P}}$, i.e., for every two vertices u and v of G it holds that whenever $F_u(T) <_T F_v(T)$ then it does not hold that $v \leq^{\mathcal{P}} u$.

▶ **Lemma 8.** *A graph G has a $\mathcal{P}$-elimination ordering of width at most ω if and only if G has a dependency tree decomposition of width at most ω. Moreover, a $\mathcal{P}$-elimination ordering of width ω can be obtained from a dependency tree decomposition of width ω in polynomial-time and vice versa.*

Proof. For the forward direction we will employ the construction given by Kloks in [19], 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 $\mathcal{P}$. Let $\leq^{\phi} = (v_1, \ldots, v_n)$ be a dependency elimination ordering for G of width ω and let H be the fill-in graph of G w.r.t. $\leq^{\phi}$. We will iteratively construct a sequence $(\mathcal{T}_0, \ldots, \mathcal{T}_{n-1})$ such that for every i with $0 \leq i < n$, $\mathcal{T}_i = (T_i, \chi_i)$ is dependency tree decompositions of the graph $H_i = H[\{v_{n-i}, \ldots, v_n\}]$ of width at most ω. Because $\mathcal{T}_{n-1}$ is a dependency tree decomposition of $H_{n-1} = H$ of width at most ω, this shows the forward direction of the lemma. In the beginning we set $\mathcal{T}_0$ 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 $0 < i < n$, $\mathcal{T}_i$ is obtained from $\mathcal{T}_{i-1}$ as follows. Note that because $N_{H_i}(v_{n-i})$ induces a clique in H_{i-1}, $\mathcal{T}_{i-1}$ contains a node that covers all vertices in $N_{H_i}(v_{n-i})$. Let t be any such bag, then is $\mathcal{T}_i$ is obtained from $\mathcal{T}_{i-1}$ 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 [19] that $\mathcal{T}_i$ satisfies the Properties (T1)–(T3) of a tree decomposition and it hence only remains to show that $\mathcal{T}_i$ satisfies (T4). Since, by induction hypothesis, $\mathcal{T}_{i-1}$ 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 and v_{n-i} for some $u \in V(H_{i-1})$. Because the only node containing v_{n-i} in $\mathcal{T}_i$ is a leaf, we can assume that $F_u(T) <_T F_{v_{n-i}}(T)$ and because $v_{n-i} \leq^{\phi} u$ it cannot hold that $v_{n-i} \leq^{\mathcal{P}} u$, as required.

For the reverse direction, let $\mathcal{T} = (T, \chi)$ be a $\mathcal{P}$-tree decomposition of G of width at most ω. It is known [19] that any linear extension of $<_{\mathcal{T}}$ is an elimination ordering for G of width at most ω. Moreover, because of Property (T4), $<_{\mathcal{T}}$ is compatible with $\leq^{\mathcal{P}}$ and hence there is a linear extension of $<_{\mathcal{T}}$, which is also a linear extension of the reverse of $\leq^{\mathcal{P}}$. ◀

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 $\mathcal{P}$) 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 $r \leq^{\mathcal{P}} c$ 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, α, β), 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:

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

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 winning 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 G is the minimum width of a winning strategy for the cop-player on G.

▶ **Lemma 9.** *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 ω can be obtained from a winning strategy for the cop-player of width $\omega + 1$ in polynomial-time and vice versa.*

Proof. Let $\mathcal{T} = (T, \chi)$ be a dependency tree decomposition of G of width ω. First, we show that $\mathcal{T}$ can be transformed into a dependency tree decomposition of width ω 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 $\chi(r) = \emptyset$ 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. By the well-known separation property of tree-decompositions, we have that $\chi(T_c) \setminus \chi(t)$ is a set of components, say containing $C_1, \dots, C_l$, of $G \setminus \chi(t)$. For every i with $1 \leq i \leq l$, let $\mathcal{T}_i = (T_i, \chi_i)$ 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 $\mathcal{T}_1, \dots, \mathcal{T}_l$ such that t now becomes adjacent to the roots $r_1, \dots, r_l$. It is straightforward to show that the result of this operation is again a dependency tree decomposition of G of width at most ω 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, α, β) where:

- $\alpha(t) = \chi(t)$ for every $t \in V(T)$,
- for a node $t \in V(T)$ with parent $p \in V(T)$, $\beta(\{p, t\}) = \chi(T_t) \setminus \chi(p)$.

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 $\mathcal{P}$-cops and robber game on G. Furthermore, it is possible to verify 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.

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

(^{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 (^{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 $\mathcal{S}$ satisfies (^{a}).

We now claim that $\mathcal{T} = (T, \alpha)$ 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 $\{u, v\} \in \alpha(t)$. Finally, in order to argue that T3 and T4 hold, we will first establish that $\mathcal{S}$ satisfies the following property:

(^{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 (^{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.

Knowing (^{b}) and (^{a}), it is not too difficult to show that T3 and T4 hold. This means that $\mathcal{T}$ is a dependency tree-decomposition, completing the proof. ◀

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 ω, determining whether a graph has dependency treewidth at most ω, and in the positive case also computing a dependency tree decomposition of width at most ω, 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.

▶ **Theorem 10.** *There is an algorithm running in time $\mathcal{O}(|V(G)|^{2\omega+2})$ that, given a graph G and a poset $\mathcal{P} = (V(G), \leq^{\mathcal{P}})$ and $\omega \in \mathbb{N}$, determines whether ω 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.*

Sketch of Proof. This algorithm is similar to the folklore $n^{\mathcal{O}(\omega)}$ algorithm for computing treewidth based on cops and robber game; see, e.g., Exercise 7.26 in Cygan et al. [7]. The idea is to transform the cops and robber game on G into a much simpler two player game, which is played on all possible positions of the cops and robber game on G.

A *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 [16]. Both players play by moving a token around on a so-called *arena*, which is a triple $\mathcal{A} = (V_C, V_R, A)$ 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 $\mathcal{A}$. 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 $\mathcal{A}$, 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 $\mathcal{A}$ 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 $\mathcal{A}$ 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 successors 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 $\mathcal{P} = (V(G), \leq^{\mathcal{P}})$, and an integer ω, we construct an arena $\mathcal{A} = (V_C, V_R, A)$ and a starting vertex $s \in V_R$ such that ω 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 $\mathcal{A}$ 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 ω 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 ω cops, and
  - $C' \subseteq V(G)$ is a potential new cop-position for at most ω cops from (C', R'), i.e., $\delta(R) \subseteq C'$ and C, R, and 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.

Finally, it is straightforward to show that the cop-player has a winning strategy from s in $\mathcal{A}$ iff G and $\mathcal{P}$ have cop-number at most ω. ◀

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 precisely compute the dependency treewidth even when this is large, and it will perform better than Theorem 10 for formulas with "tighter" dependency structures (e.g., formulas which utilize the full power of quantifier alternations).

▶ **Theorem 11.** *There is an algorithm running in time $\mathcal{O}((|V(G)|^k k^2)$ that, given a graph G and a poset $\mathcal{P} = (V(G), \leq^{\mathcal{P}})$ of width k and $\omega \in \mathbb{N}$, determines whether G and $\mathcal{P}$ admit a dependency elimination ordering of width at most ω, and if yes outputs such a dependency elimination ordering.*

Proof. To decide whether G has a dependency elimination ordering of width at most ω, 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 $|(N_G(C) \cap (D \setminus \{d\})| \leq \omega$, where C is the unique component of $G \setminus (D \setminus \{d\})$ containing d. Note that $(N_G(C) \cap (D \setminus \{d\})$ is equal to the set of neighbors of d in any fill-in graph w.r.t. to any linear order ϕ for which d is larger than all vertices in $V(G) \setminus D$ and smaller than the vertices in $D \setminus \{d\}$. Intuitively, a vertex (D, d) in H corresponds to the step in which we eliminate vertex d after exactly the vertices in $V(G) \setminus D$ have already been eliminated. 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 ω 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 ω (and output it, if it exists) in time $\mathcal{O}(|V(H)| \log(|V(H)|) + E(H))$ (e.g., by using Dijkstra's algorithm). It remains to analyze the time required to construct H (as well as its size).

Let k be the width of the poset $\mathcal{P}$. By the algorithm of Felsner, Raghavan and Spinrad [14], we can compute a chain partition $\mathcal{C} = (W_1, \ldots, W_k)$ of width k in time $\mathcal{O}(k \cdot |V(G)|^2)$. Note that every downward closed D set can be characterized by the position of the maximal element in D on each of the chains $W_1, \dots, W_k$, we obtain that there are at most $|V(G)|^k$ downward closed sets. Hence, H has at most $\mathcal{O}(|V(G)|^k(k+1))$ vertices its vertex set can be constructed in time $\mathcal{O}(|V(G)|^k(k+1))$. 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 $\mathcal{O}(|V(G)|^k k^2)$.

Hence, the total time required to construct H is $\mathcal{O}((|V(G)|^k k^2)$ 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 [30]. On the other hand, the records used in the second approach [4] do not provide sufficient information in our ordered setting.

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 *Lecture Notes in Computer Science*, 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 Víctor Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. *SIAM J. Comput.*, 45(6):2066–2086, 2016.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

8 M. Davis and H. Putnam. A computing procedure for quantification theory. 7(3):201–215, 1960.

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 Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. Small resolution proofs for QBF using dependency treewidth. *CoRR*, abs/1711.02120, 2017. `arXiv:1711.02120`.

14 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.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series.* Springer Verlag, Berlin, 2006.

16 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.

17 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.

18 Hans Kleine Büning and Theodor Lettman. *Propositional logic: deduction and algorithms.* Cambridge University Press, Cambridge, 1999.

19 T. Kloks. *Treewidth: Computations and Approximations.* Springer Verlag, Berlin, 1994.

20 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.

21 Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

22 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.

23 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.

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

25 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.

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

27 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.

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

29 Friedrich Slivovsky and Stefan Szeider. Soundness of q-resolution with dependency schemes. *Theor. Comput. Sci.*, 612:83–101, 2016.

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

31 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.

32 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.

33 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.