Tractable QBF by Knowledge Compilation

Florent Capelli
Université de Lille, Inria, UMR 9189 – CRIStAL – Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
florent.capelli@univ-lille.fr

Stefan Mengel
CNRS, CRIL UMR 8188, Lens, France
mengel@cril.fr

Abstract

We generalize several tractability results concerning the tractability of Quantified Boolean Formulas (QBF) with restricted underlying structure. To this end, we introduce a notion of width for structured DNNF which are a class of Boolean circuits heavily studied in knowledge compilation, a subarea of artificial intelligence. We then show that structured DNNF allow quantifier elimination with a size blow-up depending only on the width of the DNNF and not its size. Using known algorithms transforming restricted CNF-formulas into deterministic DNNF, we apply this result to generalize several results for counting and decision on QBF. We also complement these results with lower bounds that show that our definitions and results are essentially optimal in several senses.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases QBF, knowledge compilation, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.18

Related Version A full version of the paper is available at [7], `https://arxiv.org/abs/1807.04263`.

Funding This work was partially supported by the French Agence Nationale de la Recherche, AGGREG project reference ANR-14-CE25-0017-01.

Acknowledgements The authors would like to thank Mikaël Monet for helpful comments on an early version of this paper.

1 Introduction

It is well known that restricting the interaction between variables and clauses in CNF-formulas makes several hard problems on them tractable. For example, the propositional satisfiability problem SAT and its counting version #SAT can be solved in time $2^{O(k)}|F|$ when F is a CNF formula whose primal graph is of treewidth k [27, 24]. Many extensions of this result have been shown these last ten years for more general graph measures such as modular treewidth or cliquewidth [15, 19, 26, 23]. We here generalize in a different direction by considering decision and counting for quantified Boolean formulas (QBF) with a bounded number of quantifier alternations, i.e., we consider problems higher up in the polynomial hierarchy than SAT, resp. higher in the counting hierarchy than #SAT. It is already known that QBF as well as projected model counting, i.e., model counting for QBF with free variables and one block of existentially quantified variables, are both fixed-parameter tractable parameterized by treewidth [8, 14]. Here we generalize both these results by showing that counting the models of QBF with free variables is fixed-parameter tractable parameterized by treewidth for any bounded number of quantifier alternations. Moreover, the same is true for the strictly more general parameter of signed cliquewidth [15].

Our approach to showing these results is completely different from those used so far in the literature for treewidth restrictions of problems harder than the NP, resp. #P: we do not perform dynamic programming as e.g. in [8, 14, 12, 1]. Instead, we encode all models

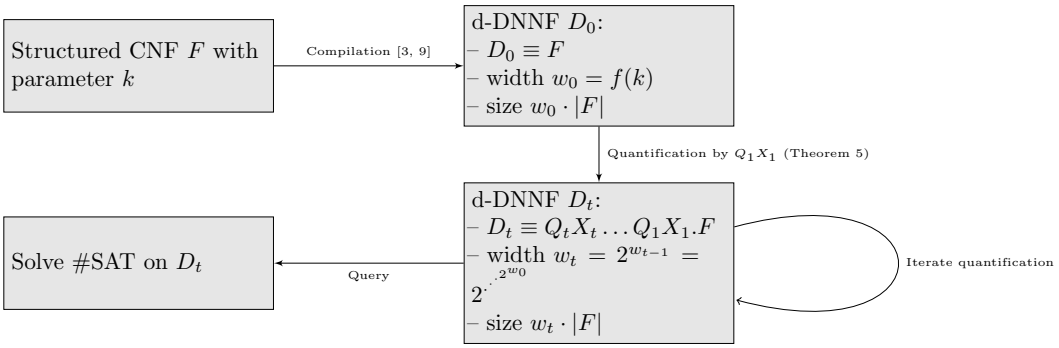


Figure 1 The overall scheme for proving tractability results on structured quantified CNF.

of the underlying CNF-formula of the given QBF into a data structure called *complete structured d-DNNF* [21], a class of circuits originating from knowledge compilation, a subarea of artificial intelligence [11]. Afterwards, we perform quantifier elimination on this data structure. When all quantifiers are eliminated, we can answer the query on the input QBF by standard algorithms for d-DNNF. Figure 1 illustrates the overall strategy.

One crucial advantage of our approach is that the first step, the transformation into d-DNNF also called *compilation*, is essentially already solved in the literature: in [3], Bova et al. recently showed that the traces of most known algorithms for structural restrictions of #SAT are essentially d-DNNF. Thus we can take these algorithms as building blocks and get the compiled representations for free without doing any additional dynamic programming.

It thus only remains to eliminate quantifiers on d-DNNF. Unfortunately, there are unconditional, exponential lower bounds showing that in general quantifier elimination on d-DNNF is impossible without blowing up the size of the representation [22]. We avoid this problem by identifying a notion of *width* for complete structured d-DNNF that is modeled after the classical width of complete OBDD. We go on to show that the size explosion during the quantifier elimination is in fact not in the size of the input but only in its width by giving a relatively simple algorithm inspired by determinization of finite automata. Since several of the compilation algorithms mentioned above yield d-DNNF whose width is independent of the input size, we get an algorithm for several restricted classes of QBF.

The resulting algorithm can be used to show that the number of models of a partially quantified CNF-formula F of treewidth k with t blocks of quantifiers can be computed in time $2^{\cdot^{\cdot^{\cdot^{2^{O(k)}}}}} |F|$ with $t+1$ exponentiations. This generalizes the result of [8] where the fixed-parameter tractability of QBF on such formulas was shown with a comparable complexity. Moreover, it generalizes the very recent result of [14] on model counting in the presence of a single existential variable block. Finally, our algorithm also applies to the more general notions of incidence treewidth and signed cliquewidth.

We complement our algorithm with lower bounds that show that our construction is essentially optimal in several respects.

The paper is organized as follows: Section 2 introduces the necessary preliminaries. Section 3 showcases our approach in a simple setting by proving that quantifier elimination can be efficiently done on small width complete OBDD. Section 4 first introduces our width notion on complete structured d-DNNF, shows some of its basic properties and then generalizes the result of Section 3 in this setting, giving our main result. The rest of the paper is dedicated to corollaries of this result proven in Section 4 and explores the limits

and optimality of our approach. Section 5 is dedicated to proving parameterized tractability results for QBF. Section 6 contains several results showing that our definition of bounded width d-DNNF cannot be weakened in several directions while still supporting efficient quantifier elimination. Finally, we close with a conclusion.

2 Preliminaries

By $\exp^\ell(p)$ we denote the iterated exponentiation function that is defined by $\exp^0(p) := p$ and $\exp^{\ell+1}(p) := 2^{\exp^\ell(p)}$.

CNF and QBF. We assume that the reader is familiar with the basics of Boolean logic and fix some notation. For a Boolean function F and a partial assignment τ to the variables of F, denote by $F[\tau]$ the function we get from F by fixing the variables of τ according to τ. For two assignments τ, σ on disjoint sets of variables we write $\tau \cup \sigma$ for the assignment on all variables of τ and σ that extends both of the assignments. A *literal* is a Boolean variable or its negation. A *clause* is a disjunction of literals and finally a *formula in conjunctive normal form* (short *CNF formula*) is a conjunction of clauses. We define the *size* $|C|$ of a clause C as the number of literals appearing in it. The size $|F|$ of a formula F is then defined as $\sum_C |C|$ where the sum is over the clauses in F.

A *Quantified Boolean Formula* (short *QBF*) $F = Q_1X_1Q_2X_2 \dots Q_\ell X_\ell \; F'$ is a CNF formula F' together with a *quantified prefix* $Q_1X_1Q_2X_2 \dots \exists X_\ell$ where $X_1, \dots, X_\ell$ are disjoint subsets of variables of F', Q_i is either $\exists$ or $\forall$ and $Q_{i+1} \neq Q_i$. The number of blocks ℓ is called the *quantifier alternation*. W.l.o.g, we assume that Q_ℓ, the most nested quantifier, is always an $\exists$-quantifier. The *quantified variables* of F are defined as $\bigcup_{i=1}^{\ell} X_i$ and the *free variables* of F are the variables of F that are not quantified. A quantified CNF naturally induces a Boolean function on its free variables.

Representations of Boolean functions. We present several representations studied in the area of knowledge compilation in a rather succinct fashion. For more details and discussion, the interested reader is referred to [11, 21].

A Boolean circuit C is defined to be in *negation normal form* (short an NNF) if $\neg$-gates appear in it only directly above the inputs. We assume that in all circuits we consider all $\wedge$-gates have exactly two inputs while all $\vee$-gates have an arbitrary positive number of inputs. An $\wedge$-gate in an NNF is called *decomposable* if, for its inputs g_1, g_2 the subcircuits rooted in g_1 and g_2 are on disjoint variable sets. A circuit in *decomposable negation normal form* (short a DNNF) is an NNF in which all gates are decomposable [9]. An $\vee$-gate g in an NNF is called *deterministic* if there is no assignment to the variables of the circuit that makes two children of g true. A DNNF is said to be *deterministic* (short a *d-DNNF*) if all its $\vee$-gates are deterministic.

A *binary decision diagram* (short *BDD*) is a directed acyclic graph with the following properties: there is one source and two sinks, one of each labeled with 0 and 1. The non-sink nodes are labeled with Boolean variables and have two outgoing edges each, one labeled with 0 the other with 1. A BDD B computes a function as follows: for every assignment a to the variables of B, one constructs a source-sink path by starting in the source and in every node labeled with a Boolean variable X following the edge labeled with $a(X)$. The label of the sink reached this way is then the value computed by B on a.

A BDD is called a *free* BDD (short *FBDD*) if on every source-sink path every variable appears at most once. If on every path the variables are seen in a fixed order π, then the FBDD is called an ordered BDD (short *OBDD*).

An FBDD is called *complete* if on every source-sink path every variable appears exactly once. This notion also applies to OBDD in the obvious way. A *layer* of a variable X in a complete OBDD B is the set of all nodes labeled with X. The *width* of B is the maximum size of its layers. Note that for every OBDD one can construct a complete OBDD computing the same function in polynomial time, but it is known that it is in general unavoidable to increase the number of nodes labeled by a variable by a factor linear in the number of variables [2].

For any representation D of a Boolean function in one of the above forms, we denote by $\mathsf{var}(D)$ the set of variables appearing in D.

Graphs of CNF formulas. There are two graphs commonly assigned to CNF formulas: the *primal graph* of a CNF formula F is the graph that has as its vertices the variables of F and there is an edge between two vertices x, y if and only if there is clause in F where both variables x and y appear. The *incidence graph* of F has as vertices the variables and the clauses of F and there is an edge between two nodes x and C if and only if x is a variable, C is a clause, and x appears in C.

We will consider several width measures on graphs like treewidth and pathwidth. Since we do not actually need the definitions of these measures but only depend on known results on them, we spare the readers these rather technical definitions and give pointers to the literature in the respective places.

3 Warm-up: Quantification on OBDD

In this section, we will illustrate the main ideas of our approach on the simpler case of OBDD. To this end, fix an OBDD G in variables $x_1, \ldots, x_n$ in that order. Now let Z be a set of variables. We want to compute an OBDD that encodes $\exists Z\, G$, i.e., we want to forget the variables in Z.

Note that it is well-known that OBDD do not allow arbitrary forgetting of variables without an exponential blow-up, see [11]. Here we make the observation that this exponential blow-up is in fact not in the *size* of the considered OBDD but in the *width* which for many interesting cases is far lower.

▶ **Lemma 1.** *Let G be a complete OBDD of width w and Z be a subset of its variables. Then there is an OBDD of width at most 2^w that computes the function of $\exists Z\, G$.*

Proof. The technique is essentially the power set construction used in the determinization of finite automata. Let V_x for a variable x denotes the set of nodes labeled by x. For every x not in Z, our new OBDD G' will have a node N_S labeled by x for every subset $S \subseteq V_x$. The invariant during the construction will be that a partial assignment a to the variables in $\mathsf{var}(G) \setminus Z$ that come before x in G leads to N_S if and only if S is the set of nodes in V_x which can be reached from the source by an extension of a on the variables of Z. We make the same construction for the 0- and 1-sink of G: G' gets three sinks 0, 1 and 01 which encode which sinks of G can be reached with extensions of an assignment a. Note that if we can construct such a G', we are done by merging the sinks 1 and 01.

The construction of G' is fairly straightforward: consider a variable x not in Z and let x' be the next variable not in Z. For every node $N \in V_x$, we compute the set of nodes N^+ labeled with x' that we can reach by following the 1-edge of N and the set of N^- nodes labeled with x' that we can reach by following the 0-edge of N. Then, for every $S \subseteq V_x$ we define the 1-successor of N_S as $N_{S'}$ where $S' = \bigcup_{N \in S} N^+$. The 0-successors are defined analogously. ◀

We remark that in [13] a related result is shown: for a CNF-formula F of pathwidth k and every subset Z of variables, one can construct an OBDD of size $2^{2^k}|F|$ computing $\exists Z\ F$. This result follows easily from Lemma 1 by noting that for a CNF F of pathwidth k one can construct a complete OBDD of width 2^k. We note that our approach is more flexible than the result in [13] because we can iteratively add more quantifier blocks since $\forall Z\ D \equiv \neg(\exists Z \neg D)$ and negation in OBDD can be easily performed without size increase. For example, one directly gets the following corollary.

▶ **Corollary 2.** *There is an algorithm that, given a QBF restricted to ℓ quantifier alternations and of pathwidth k, decides if F is true in time $O(\exp^{\ell}(p)|F|)$.*

Note that Corollary 2 is already known as it is a special case of the corresponding result for treewidth in [8]. However, we will show that a similar approach to that of Lemma 1 can be used to derive several generalizations of the result of [8]: we show that we can add quantification to bounded width structured d-DNNF, a generalization of OBDD (see Section 4). Since several classes of CNF formulas are known to yield bounded width structured d-DNNF [3], this directly yields QBF algorithms for these classes, see Section 5 for details.

4 Bounded width complete structured DNNF

Before formulating and proving our main result, we first introduce our central data structure called *complete structured DNNF* as a generalization of OBDD and a restriction of the structured DNNF from [21]. We introduce a width notion for it and show how to deal with constants in the setting. After these preparations, we then show our main result on eliminating quantifiers in Section 4.4.

4.1 Complete structured DNNF

A *vtree* T for a set of variables X is a rooted tree where every non-leaf node has exactly two children and the leaves of T are in one-to-one correspondence with X. A *complete structured* DNNF (D, T, λ) is a DNNF D together with a vtree T for $\mathsf{var}(D)$ and a labeling λ of the nodes of T with sets of gates of D such that:

- If t is a leaf of T labeled with variable $x \in X$ then $\lambda(t)$ contains only input gates of D labeled with either x, $\neg x$.
- For every gate u of D, there exists a unique node t_u of T such that $u \in \lambda(t_u)$.
- There is no non-leaf node t of T such that $\lambda(t)$ contains an input gate of D.
- For every $\wedge$-gate u with inputs v_1, v_2, we have $t_{v_1} \neq t_{v_2}$.
- For every edge (u, v) of D:
  - Either v is an $\wedge$-gate, u is an $\vee$-gate or an input gate and t_u is the child of t_v.
  - Or v is an $\vee$-gate, u is an $\wedge$-gate and $t_u = t_v$.

Intuitively, T can be seen as a coarse structure of D, as depicted on Figure 2: We structure the gates of D into blocks $\lambda(t)$ that are associated to nodes t of T. Every such $\lambda(t)$ computes a 2DNF where every term has one input from $\lambda(t_1)$ and $\lambda(t_2)$, respectively, where t_1, t_2 are the children of t. In the following, when we do not directly deal with vtree and its labeling, we may refer to a complete structured DNNF (D, T, λ) by only mentioning the circuit D. It is then always understood that T and λ with the desired properties exist.

We note that there is a syntactic transformation of complete OBDD into complete structured d-DNNF. It proceeds iteratively from the sinks to the source introducing a gate g_v for every node v: for every sink, g_v is an input gate with the same label as the sink. For every other node v with label x, a 0-edge to u and a 1-edge to u', we introduce a subcircuit

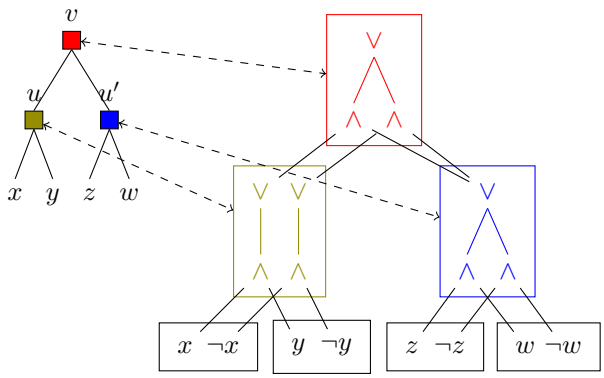


Figure 2 A vtree T and a complete structured DNNF (D, T, λ), where λ for the nodes v, u, w is represented with colors and dashed arrows.

computing $(g_u \wedge \neg x) \vee (g_{u'} \wedge x)$. It is easy to check that the resulting circuit is a structured complete d-DNNF computing the right function and whose vtree consists of a tree in which for every internal node one of the children is a leaf[1].

4.2 Width

We define the *width* of a complete structured DNNF (D, T, λ) as $\max_{t \in V(T)} |\{v \in \lambda(t) \mid v \text{ is an } \vee\text{-gate}\}|$. For example, the DNNF pictured on Figure 2 has width 2 since $\lambda(u)$ contains 2 $\vee$-gates and $\lambda(u')$ and $\lambda(v)$ contain less $\vee$-gates.

Note that for the width we do not take into account $\wedge$-gates. This is for several reasons: first, only considering $\vee$-gates simplifies some of the arguments later on and gives cleaner results and proofs. Moreover, it is not hard to see that when rewriting OBDD as DNNF as sketched above, the width of the original OBDD is exactly the width of the resulting circuit. The same is also true for the width of SDD [5], another important representation of Boolean functions [10]. Thus, width defined only on $\vee$-gates allows a tighter connection to the literature. Finally, the following observation shows that the number of $\wedge$-gates in a complete structured DNNF as we define it is highly connected to the width.

▶ **Observation 3.** *Let (D, T, λ) be a complete structured DNNF of width $w \geq 2$. We can in linear time in $|D|$ compute a complete structured DNNF (D', T, λ') of width w and equivalent to D such for every node t of T, we have $|\lambda'(t)| \leq (w^2 + w)$. Moreover, D' is of size at most $2(w + w^2)|\mathsf{var}(D)|$.*

Proof. For the first statement, note that by definition there are at most w $\vee$-gates in $\lambda(t)$. Now, the inputs of every $\wedge$-gate of $\lambda(t)$ are either $\vee$-gates or input gates in $\lambda(t_1)$ and $\lambda(t_2)$ where t_1, t_2 are the children of t in T. Thus, there are at most w^2 possible ways of connecting these $\wedge$-gates to their inputs. So if we eliminate $\wedge$-gates that have identical inputs and keep for every combination at most one of them, we get D' with the desired size bound on $\lambda'(t)$. However, we can neither naively compare the children of all $\wedge$-gates nor order the $\wedge$-gates by their children to eliminate $\wedge$-gates with identical inputs since both approaches would violate the linear time requirement.

[1] Note that strictly speaking the constructed circuit is not a complete structured d-DNNF as defined above because it contains constants as input gates. This slight complication will be taken care of in Lemma 4 below.

To avoid this slight complication, we proceed as follows: in a first step, we count the $\wedge$-gates in $\lambda(t)$. If there are at most w^2 of them, we satisfy the required upper bound, so we do nothing. Otherwise, we create an array of size w^2 indexed by the pairs of potential inputs of $\wedge$-gates in $\lambda(t)$. We initialize all cells to some null-value. Now we iterate over the $\wedge$-gates in $\lambda(t)$ and do the following for every such gate u: if the cell indexed by the children of u is empty, we store u in that cell and continue. If there is already a gate u' in the cell, we connect all gates that u feeds into to u' and delete u afterwards. It is easy to see that the resulting algorithm runs in linear time, computes a D' equivalent to D and satisfies the size bounds on $\lambda(t)$.

Since T is a tree where every node but the leaves has exactly 2 children, the number of nodes in T is at most $2|\mathsf{var}(D)|$. Now, because of $|\lambda'(t)| \leq w^2 + w$ for every t in T, the bound on $|D'|$ follows directly. ◀

We remark that complete structured DNNF as defined above are more restrictive than structured DNNF as defined in [21]. The definition of [21] only gives a condition on the way decomposable $\wedge$-gates can partition variables, following the vtree. However, it is possible to to extend the classical construction to turn any OBBD into a complete one to transform any structured DNNF in the sense of [21] into the form we define above with only a polynomial increase in size. However, even for OBDD this rewriting may increase the width arbitrarily at least when not changing the order [2]. Moreover, the construction for structured DNNF is rather tedious and complicated, so we will not follow this direction here.

4.3 Eliminating Constants

Our definition of complete structured DNNF does not allow constant inputs. This is in general not a problem as constants can be propagated in the circuits and thus eliminated. However, it is not directly clear how this propagation affects the width in our setting. Moreover, most of our algorithms are easier to describe by allowing constants. So let us spend some time to deal with constants in our setting. To this end, we introduce the notion of *extended vtrees*. An extended vtree T on a variable set X is defined as a vtree in which we allow some leaves to be unlabeled. Every variable of X must be the label of exactly one leaf still. A complete structured DNNF (D, T, λ) is defined as for an extended vtree with the additional requirement that for every unlabeled leaf ℓ of T, $\lambda(\ell)$ is a set of constant inputs of D.

We now show that we can remove the unlabeled leaves without increasing the width.

▶ **Lemma 4.** *There is a linear time algorithm that, given a complete structured DNNF (resp. d-DNNF) (D, T, λ) of width w where T is an extended vtree, computes a complete structured DNNF (resp. d-DNNF) (D', T', λ') of width w that is equivalent to D where T' is non-extended.*

Proof. Given an extended vtree T and a leaf ℓ, let $T \setminus \ell$ be the vtree obtained by removing the leaf ℓ of T and by merging the father and the sibling of ℓ in T. We first show that there is an algorithm that, given a complete structured DNNF (resp. d-DNNF) (D, T, λ) of width w and a non-labeled leaf ℓ of T, computes in linear time in $|\lambda(t)|$ an equivalent complete structured DNNF $(D', T \setminus \ell, \lambda')$ of width at most w. Iterating the construction and observing that every $\lambda(t)$ is treated only once, we get the claim of the lemma.

Let t be the father and t_s the sibling of ℓ in T. We let t' be the vertex of $T \setminus \ell$ obtained by merging t and t_s (see Figure 3). The idea of the transformation is depicted on Figures 4.

By definition, all gates of $\lambda(t)$ that are connected to gates in $\lambda(\ell)$ are $\wedge$-gates. We remove every $\wedge$-gate of $\lambda(t)$ connected to constant 0 as they are equivalent to 0 and are connected

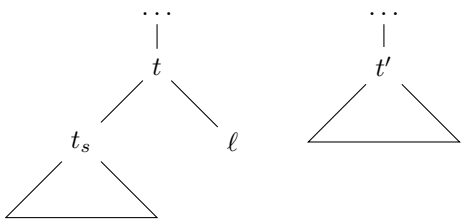


Figure 3 The trees T and $T \setminus \ell$ with notations.

to $\vee$-gates of $\lambda(t)$. We next deal with the $\wedge$-gates of $\lambda(t)$ connected to the constant 1. For every such gate v, we connect its other input to all outputs of v which by definition are all $\vee$-gates in $\lambda(t)$. This does not change the functions computed by the outputs of v and does not affect the determinism of the DNNF.

If t_s is not a leaf of T, then $\vee$-gates of $\lambda(t)$ are connected to $\vee$-gates of $\lambda(t_s)$. Without changing the function computed nor determinism, we can connect the $\vee$-gates of $\lambda(t)$ directly to the input of its inputs and thus remove every $\vee$-gate of $\lambda(t_s)$. Now the circuit has the following form: $\vee$-gates of $\lambda(t)$ are connected to $\wedge$-gates of $\lambda(t_s)$. We thus define $\lambda'(t')$ as the remaining $\vee$-gates of $\lambda(t)$ and $\wedge$-gates of $\lambda(t_s)$ and get a complete structured DNNF for $T \setminus \ell$. The number of $\vee$-gates in $\lambda(t')$ is less than in $\lambda(t)$ so the width has not increased.

If t_s is a leaf labeled by a variable x, then every $\vee$-gate g in $\lambda(t)$ is connected to input gates in x and thus they compute either x, or $\neg x$ or $\neg x \vee x$. In the former two cases, we simply substitute g by x or $\neg x$ respectively. If g computes $\neg x \vee x$, then do for every $\wedge$-gate g' that has g as an input the following: create a clone g'' of g', i.e., a new $\wedge$-gate that has the same inputs and outputs as g'. Then substitute the input g of g' by $\neg x$ and by x for g''. Since all gates that have g' as an input are $\vee$-gates, this does not change the function computed in these values. Finally, delete g. Doing this for all g, we delete all $\wedge$-gates in $\lambda(t)$. Now setting $\lambda'(t)$ to contain the newly introduced input gates completes the construction. Obviously, the number of $\vee$-gates in $\lambda'(t^*)$ is never bigger than that in $\lambda(t^*)$.

Finally, if t_s is an unlabeled leaf, then all $\vee$-gates in $\lambda(t)$ compute constants. Substituting them by those constants and defining $\lambda'(T')$ in the obvious way, completes the the proof. ◀

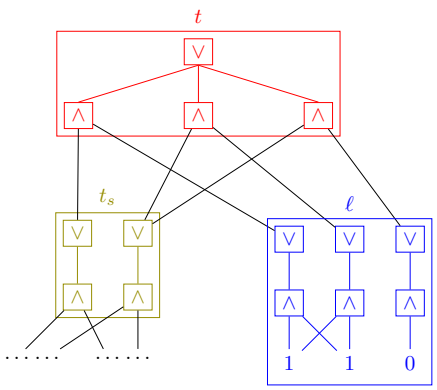
4.4 Existential quantification on bounded width d-DNNF

In this section, we give an algorithm that allows us to quantify variables in d-DNNF. The main result is the following.

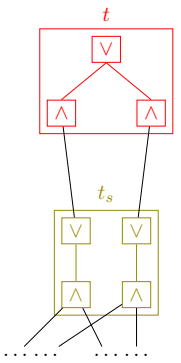
▶ **Theorem 5.** *There is an algorithm that, given a complete structured DNNF (D, T, λ) of width w and $Z \subseteq \mathsf{var}(D)$, computes in time $2^{O(w)}|D|$ a complete structured d-DNNF (D', T', λ') of width at most 2^w having a designated gate computing $\exists Z\ D$ and another designated gate computing $\neg\exists Z\ D$.*

In the rest of this section, we will prove Theorem 5. Let (D, T, λ) be a complete structured DNNF. Let $X = \mathsf{var}(D)$, the variables $Z \subseteq X$ those that we will quantify and w the width of D.

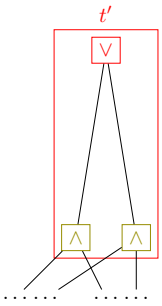
Given a node t of T, let $\mathsf{var}(t)$ be the set of variables which are at the leaves of the subtree of T rooted in t. We define $\mathsf{forgot}(t) := Z \cap \mathsf{var}(t)$ and $\mathsf{kept}(t) := \mathsf{var}(t) \setminus \mathsf{forgot}(t)$. Intuitively, $\mathsf{forgot}(t)$ contains the set of variables that are quantified away below t while $\mathsf{kept}(t)$ contains the remaining variables under t. Let D_v for a gate v denote the sub-DNNF of D rooted in v.



(a) Original circuit.



(b) Propagating constants. The rightmost part evaluates to 0 and disappears.



(c) Merging t_s and t and cleaning disconnected gates.

Figure 4 Illustration of the transformation of Lemma 4. The constants are propagated in the first step to remove gates in bag ℓ. Then the bags of t_s and t are merged without changing the computed function.

Shapes. A key notion for our algorithm will be what we call *shapes*. Let t be a node of T and let O_t be the set of $\vee$-gates of D labeling t. An assignment $\tau : \mathsf{kept}(t) \to \{0,1\}$ is of shape $S \subseteq O_t$ if and only if

$$S = \{s \in O_t \mid \exists \sigma : \mathsf{forgot}(t) \to \{0,1\}, \tau \cup \sigma \models D_s\}.$$

We denote by $\mathsf{Shape}_t \subseteq 2^{O_t}$ the set of shapes of a node t. Observe that $|\mathsf{Shape}_t| \leq 2^{|O_t|} \leq 2^w$ since $|O_t| \leq w$ by definition.

The key observation is that Shape_t can be inductively computed. Indeed, let t be a node of T with children t_1, t_2 and let $S_1 \in \mathsf{Shape}_{t_1}$, $S_2 \in \mathsf{Shape}_{t_2}$. We define $S_1 \bowtie S_2 \subseteq O_t$ to be the set of gates $s \in O_t$ that evaluate to 1 once we replace every gate in S_1 and S_2 by 1 and every gate in $O_{t_1} \setminus S_1$ and $O_{t_2} \setminus S_2$ by 0.

▶ **Lemma 6.** *Let t be node of T with children t_1, t_2. Let $\tau_1 : \mathsf{kept}(t_1) \to \{0,1\}$ be of shape S_1 and $\tau_2 : \mathsf{kept}(t_2) \to \{0,1\}$ be of shape S_2. Then $\tau = \tau_1 \cup \tau_2$ is of shape $S_1 \bowtie S_2$.*

Proof. Let S be the shape of τ. We first prove $S \subseteq S_1 \bowtie S_2$. So let $s \in S$. Since τ is of shape S, there exists $\sigma : \mathsf{forgot}(t) \to \{0,1\}$ such that $\tau \cup \sigma$ satisfies D_s. Since s is an $\vee$-gate, there must be an input gate s' of s such that $\tau \cup \sigma$ satisfies s'. By definition, s' is an $\wedge$-gate with two children $s_1 \in O_{t_1}$ and $s_2 \in O_{t_2}$. Thus D_{s_1} is satisfied by $(\tau \cup \sigma)|_{\mathsf{var}(t_1)} = \tau_1 \cup \sigma|_{\mathsf{var}(t_1)}$. Consequently, $s_1 \in S_1$ since S_1 is the shape of τ_1. Similarly $s_2 \in S_2$. Thus, in the construction of $S_1 \bowtie S_2$, both s_1 and s_2 are replaced by 1, so s evaluates to 1, that is, $s \in S_1 \bowtie S_2$.

We now show that $S_1 \bowtie S_2 \subseteq S$. So let $s \in S_1 \bowtie S_2$. Then, in the construction of $S_1 \bowtie S_2$, there must be an input s' of s that is satisfied. Then s' is an $\wedge$-gate with children $s_1 \in O_{t_1}$, $s_2 \in O_{t_2}$ evaluating to 1. It follows that s_1 and s_2 have been replaced by 1 in the construction of $S_1 \bowtie S_2$. Now by definition of S_1, there exists $\sigma_1 \colon \mathsf{forgot}(t_1) \to \{0,1\}$ such that $\tau_1 \cup \sigma_1$ satisfies D_{s_1} and $\sigma_2 \colon \mathsf{forgot}(t_2) \to \{0,1\}$ such that $\tau_2 \cup \sigma_2$ satisfies D_{s_2}. Thus, $(\tau_1 \cup \sigma_1) \cup (\tau_2 \cup \sigma_2) = \tau \cup (\sigma_1 \cup \sigma_2)$ is well-defined because σ_1 and σ_2 do not share any variables because s' is decomposable. Moreover, $\tau \cup (\sigma_1 \cup \sigma_2)$ satisfies D_s and thus we have $s \in S$. ◀

Constructing the projected d-DNNF. We now inductively construct a d-DNNF D' computing $\exists Z\ D$ and of width at most 2^w. The extended vtree T' for D' is obtained from T by removing the labels of the leaves corresponding to variables in Z. One can then apply Lemma 4 to obtain a vtree. We inductively construct for every node t of T and $S \in \mathsf{Shape}_t$, an $\vee$-gate $v_t(S)$ in D' such that $D'_{v_t(S)}$ accepts exactly the assignments of shape S and we will define $\lambda'(t) = \bigcup_{S \in \mathsf{Shape}_t} v_t(S)$.

If t is a leaf of T, then $\mathsf{kept}(t)$ has at most one variable, thus we have at most two assignments of the form $\mathsf{kept}(t) \to \{0,1\}$. We can thus try all possible assignments to compute Shape_t explicitly and $v_t(S)$ will either be a literal or a constant for each $S \in \mathsf{Shape}_t$. We put $v_t(S)$ in $\lambda'(t')$ where t' is the leaf of T' corresponding to t. It is clear that if t' is labeled with variable x then $v_t(S)$ is a literal labeled by x or by $\neg x$. If t' is unlabeled, then it corresponds to a leaf t of T labeled with a variable of Z. Thus $v_t(S)$ is a constant input so the conditions of structuredness are respected.

Now let t be a node of T with children t_1, t_2 and assume that we have constructed $v_{t_1}(S_1)$ for every $S_1 \in \mathsf{Shape}_{t_1}$ and $v_{t_2}(S_2)$ for every $S_2 \in \mathsf{Shape}_{t_2}$. We define $v_t(S)$ as:

$$\bigvee_{S_1,S_2 : S = S_1 \bowtie S_2} v_{t_1}(S_1) \wedge v_{t_2}(S_2)$$

where S_1, S_2 run over Shape_{t_1} and Shape_{t_2} respectively.

First of all, observe that the $\wedge$-gates above are decomposable since $D'_{v_{t_1}(S_1)}$ is on variables $\mathsf{kept}(t_1)$ which is disjoint from $\mathsf{kept}(t_2)$, the variables of $D'_{v_{t_2}(S_2)}$.

Moreover, observe that the disjunction is deterministic. Indeed, by induction, τ satisfies the term $v_{t_1}(S_1) \wedge v_{t_2}(S_2)$ if and only if $\tau|_{\mathsf{var}(t_1)}$ is of shape S_1 and $\tau|_{\mathsf{var}(t_2)}$ is of shape S_2. Since an assignment has exactly one shape, we know that τ cannot satisfy another term of the disjunction.

Finally, we have to show that $v_t(S)$ indeed computes the assignments of shape S. This is a consequence of Lemma 6. Indeed, if τ is of shape S then let S_1, S_2 be the shapes of $\tau|_{\mathsf{var}(t_1)}$ and $\tau|_{\mathsf{var}(t_2)}$ respectively. By Lemma 6, $S = S_1 \bowtie S_2$ and then $\tau \models v_{t_1}(S_1) \wedge v_{t_2}(S_2)$, and then, $\tau \models v_t(S)$.

Now, if $\tau \models v_{t_1}(S_1) \wedge v_{t_2}(S_2)$ for some S_1 and S_2 in the disjunction, then we have by induction that $\tau|_{\mathsf{var}(t_1)}$ and $\tau|_{\mathsf{var}(t_2)}$ are of shape S_1 and S_2 respectively. By Lemma 6, τ is of shape $S_1 \bowtie S_2 = S$.

Let t' be the node of T' corresponding to t. We put all gates needed to compute $v_t(S)$ in $\lambda'(t')$ for every S. This has the desired form: a level of $\vee$-gate, followed by a level of $\wedge$-gate connected to $\vee$-gates in $\lambda'(t'_1)$ and $\lambda'(t'_2)$. By construction, the width of the d-DNNF constructed so far is $\max_t |\mathsf{Shape}_t| \leq 2^w$.

Now assume that we have a d-DNNF D_0 with a gate $v_t(S)$ for every t and every $S \in \mathsf{Shape}_t$ computing the assignments of shape τ. Let r be the root of T. We assume w.l.o.g. that the root of D is a single $\vee$-gate r_o connected to every $\wedge$-gate labeled by r. Then $v_r(\{r_o\})$ accepts exactly $\exists Z D$ and $v_r(\emptyset)$ accepts $\neg\exists Z\ D$.

5 Algorithms for graph width measures

In this section, we will show how we can use the result of Section 4 in combination with known compilation algorithms to show tractability results for QBF with restricted underlying graph structure and bounded quantifier alternation. This generalizes the results of [8, 13, 14].

We use the following result which can be verified by careful analysis of the construction in [9, Section 3]; for the convenience of the reader we give an independent proof in the long version of this paper [7].

▶ **Theorem 7.** *There is an algorithm that, given a CNF F of primal treewidth k, computes in time $2^{O(k)}|F|$ a complete structured d-DNNF D of width $2^{O(k)}$ equivalent to F.*

We lift Theorem 7 to incidence treewidth by using the following result from [17].

▶ **Proposition 8.** *There is an algorithm that, given a CNF-formula F of incidence treewidth k, computes in time $O(2^k|F|)$ a 3CNF-formula F' of primal treewidth $O(k)$ and a subset Z of variables such that $F \equiv \exists Z F'$.*

▶ **Corollary 9.** *There is an algorithm that, given a CNF F formula of incidence treewidth k, computes in time $2^{O(k)}|F|$ a complete structured d-DNNF D of width $2^{O(k)}$ and a subset Z of variables such that $F \equiv \exists Z D$.*

Note that in [3] there is another algorithm that compiles bounded incidence treewidth into d-DNNF without introducing new variables that have to be projected away to get the original function. The disadvantage of this algorithm though is that the time to compile is quadratic in the size of F. Since we are mostly interested in QBF in which the last quantifier block is existential, adding some more existential variables does not hurt our approach, so we opted for the linear time algorithm we get from Corollary 9.

Now using Theorem 5 iteratively, we directly get the following result.

▶ **Theorem 10.** *There is an algorithm that, given a QBF F with free variables, ℓ quantifier blocks and of incidence treewidth k, computes in time $\exp^{\ell+1}(O(k))|F|$ a complete structured d-DNNF of width $\exp^{\ell+1}(O(k))$ accepting exactly the models of F.*

Proof. Let $F = Q_1 X_1 \dots \exists X_\ell G$. We use Corollary 9 to construct a structured DNNF D of width $2^{O(k)}$ such that $G \equiv \exists Z D$, that is $F \equiv Q_1 X_1 \dots \exists (X_\ell \cup Z) D$. Assume first that $Q_1 = \forall$. Then using the fact that for every formula F' we have $\forall X F' \equiv \neg\exists\neg F'$, we can rewrite this into $F \equiv \neg\exists X_1(\neg\exists X_2(\neg \dots (\neg\exists(X_\ell \cup Z)G)\dots))$. We now use Theorem 5 to iteratively from the right eliminate all blocks $\neg\exists X_i$. The result is a complete structured d-DNNF accepting exactly the models of F. If $Q_1 = \exists$, we apply essentially the same construction with the only difference that there is no negation in front of the formula which can be also dealt with using Theorem 5. Each application of Theorem 5 blows the width of the circuit by a single exponential, resulting in the stated complexity. ◀

As an application of Theorem 10, we give a result on model counting.

▶ **Corollary 11.** *There is an algorithm that, given a QBF F with free variables, ℓ quantifier blocks and of incidence treewidth k, computes in time $\exp^{\ell+1}(O(k))|F|$ the number of models of F.*

We remark that Corollary 11 generalizes several results from the literature. On the one hand, it generalizes the main result of [8] from decision to counting, from primal treewidth to incidence treewidth and gives more concrete runtime bounds[2]. On the other hand, it generalizes the counting result of [14] from projected model counting, i.e., QBF formulas with free variables and just one existential quantifier block, to any constant number of quantifier alternations. Moreover, our runtime is linear in $|F|$ in contrast to the runtime of [14] which is quadratic.

As a generalization of Theorem 10, let us remark that there are compilation algorithms for graph measures beyond treewidth. For example, it is known that CNF formulas of *bounded signed cliquewidth* [15] can be compiled efficiently [3]. More exactly, there is an algorithm

[2] We remark that the latter two points have already been made recently in [17].

that compiles a CNF formula F of signed incidence cliquewidth k in time $2^{O(k)}|F|^2$ into a structured d-DNNF of size $2^{O(k)}|F|$. We will not formally introduce signed incidence cliquewidth here but refer the reader to [15, 6]. Inspecting the proof of [3], one can observe that the algorithm construct a complete structured d-DNNF of width at most $2^{O(k)}$ which as above yields the following result.

▶ **Theorem 12.** *There is an algorithm that, given a QBF F with free variables, with ℓ quantifier blocks and of signed incidence cliquewidth k, computes in time $\exp^{\ell+1}(O(k))|F| + 2^{O(k)}|F|^2$ a complete structured d-DNNF of width $\exp^{\ell+1}(O(k))$ accepting exactly the models of F.*

With Theorem 12 it is now an easy exercise to derive generalizations of [8, 14, 15].

In the light of the above positive results one may wonder if our approach can be pushed to more general graph width measures that have been studied for propositional satisfiability like for example modular treewidth [20] or (unsigned) cliquewidth [26]. Using the results of [18], we can answer this question negatively in two different ways: on the one hand, QBF of bounded modular cliquewidth and bounded incidence cliquewidth with one quantifier alternation is NP-hard, so under standard assumptions there is no version of Theorem 10 and thus also not of Corollary 9 for cliquewidth. On the other hand, analyzing the proofs of [18], one sees that in fact there it is shown that for every CNF formula F there is a bounded modular treewidth and bounded incidence cliquewidth formula F' and a set Z of variables such that $F \equiv \exists Z F'$. Since it is known that there are CNF formulas that do not have subexponential size DNNF [4], it follows that there are such formulas F' such that every DNNF representation of $\exists Z F'$ has exponential width. This unconditionally rules out a version of Corollary 9 and Theorem 12 for modular treewidth or cliquewidth.

6 Lower Bounds

In this section, we will show that all restrictions we put onto the DNNF in Theorem 5 are necessary.

6.1 The definition of width

Width of an OBDD is usually defined on *complete* OBDD. There is however another way of defining width for OBDD by just counting the number of nodes that are labeled with the same variable which for a non-complete OBDD might be far smaller. Let us call this notion *weak width*. We will show that width in Theorem 5 cannot be substituted by weak width.

▶ **Lemma 13.** *For every n there is an OBDD D_n in $O(n)$ variables of weak width 3 and a set $Z \subseteq \mathsf{var}(D_n)$ such that $\neg\exists Z\ D_n$ does not have an DNNF of size $2^{o(n)}$.*

Proof. Let S_i for $i \in \mathbb{N}$ denote the term $\neg z_i \wedge \left(\bigwedge_{j\in[i-1]} z_j\right)$. For a CNF $F = C_1 \wedge \ldots \wedge C_m$ we then define the function

$$F' = \bigvee_{i=1}^{m} S_i \wedge C_i.$$

It is easy to see that by testing $z_1, \ldots, z_m$ successively and branching a small OBDD for C_i at each 0-output of the decision node testing z_i as depicted on Figure 5, one can construct an OBDD of size $O(|F|)$ computing F'. If every variable appears in at most three clauses of F, then this OBDD has weak width 3 since a variable x is only tested for clauses where it appears.

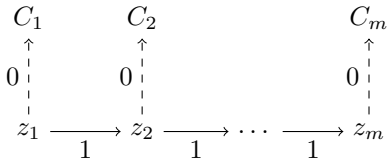


Figure 5 Structure of an OBDD for F'.

Note that $\forall Z\ F' \equiv F$. Since there are CNF formulas in which every variable appears in at most three clauses which do not have subexponential size DNNF [4]. It follows that for such F the function $\forall Z\ F'$ has exponential size. Now remarking that $\forall Z\ F' \equiv \neg\exists Z\ (\neg F')$ and that $\neg F'$ has an OBDD of weak width 3 as well, completes the proof. ◀

6.2 Structuredness

One of the properties required for Theorem 5 is that we need the input to be structured. Since structuredness is quite restrictive, see e.g. [22], it would be preferable to get rid of it to show similar results. Unfortunately, there is no such result as the following lemma shows.

To formulate our results, we need a definition of width for FBDD. This is because width as we have defined it before depends on the vtree of the DNNF which we do not have in the case without structuredness. To define width for the unstructured case, we consider layered FBDD: an FBDD F is called *layered* if the nodes of F can be partitioned into sets $L_1, \dots, L_s$ such that for every edge uv in F there is an $i \in [s]$ such that $u \in L_i$ and $v \in L_{i+1}$. The *width* of F is then defined as $\max\{|L_i| \mid i \in [s]\}$.

▶ **Lemma 14.** *For every n there is a function f_n in $O(n^2)$ variables with an FBDD representation of size $O(n^2)$ and width $O(1)$ such that there is a variable x of f_n such that every deterministic DNNF for $\exists x\ f_n$ has size $2^{\Omega(n)}$.*

Proof. We use a function introduced by Sauerhoff [25]: let $g : \{0,1\}^n \to \{0,1\}$ be the function that evaluates to 1 if and only if the sum of its inputs is divisible by 3. For a $n \times n$-matrix X with inputs $x_{ij} \in \{0,1\}$, we define

$$R_n(X) := \bigoplus_{i=1}^{n} g(x_{i1}, x_{i2}, \dots, x_{in})$$

where $\oplus$ denotes addition modulo 2 and define $C_n(X) := R_n(X^T)$ where X^T is the transpose of X. Then $S_n(X) := R_n(X) \vee C_n(X)$.

Note that, ordering the variables of X by rows, resp. columns, R_n and C_n both have OBDD of width $O(1)$ and size $O(n^2)$. Now let $S'_n = (x \wedge R_n) \vee (\neg x \wedge C_n)$. Then S'_n clearly has an FBDD of size $O(n^2)$ and width $O(1)$: decide on x first and then depending on its value follow the OBDD for R_n or C_n.

But $\exists x S'_n(X) = S_n(X)$ which completes the proof since S_n is known to require size $2^{\Omega(n)}$ for deterministic DNNF [4]. ◀

7 Conclusion

We have introduced a new notion of width of complete structured d-DNNF and shown that using it, in combination with a rather simple quantifier elimination result and known compilation results, one can show several new tractability results around QBF. In contrast

to earlier results that solved similar problems in one pass of dynamic programming [8, 14], our approach is iterative and only considers one quantifier block at the same time which in our opinion greatly simplifies the argument. Moreover, factoring out the initial compilation phase allowed us to generalize known results for treewidth to signed cliquewidth essentially for free.

We feel that the notion of width we introduced for complete structured d-DNNF is an interesting notion, independent of the results on quantifier elimination here, and deserves closer examination. In the long version of this paper [7] we initiate this by giving results on standard transformation that are generally considered in knowledge compilation. We are certain that beyond this our width notion will have more uses in the future.

It might be interesting to apply our approach to non-monotone reasoning problems from artificial intelligence. There is a wealth of such problems that have been considered under treewidth reductions, see e.g. the problems in [16], but for more general width measures far less is known, some exceptions being [12, 1]. Proving more such results might be possible by reductions to QBF as in [17] but it would be necessary to understand if those reductions maintain the considered width measure. Alternatively, one could try to mimic our approach of compiling and then refining the solution space iteratively for the individual problems which might be easier than performing direct dynamic programming.

References

1 Bernhard Bliem, Sebastian Ordyniak, and Stefan Woltran. Clique-Width and Directed Width Measures for Answer-Set Programming. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1105–1113. IOS Press, 2016. doi:10.3233/978-1-61499-672-9-1105.

2 Beate Bollig and Ingo Wegener. Asymptotically Optimal Bounds for OBDDs and the Solution of Some Basic OBDD Problems. *J. Comput. Syst. Sci.*, 61(3):558–579, 2000. doi:10.1006/jcss.2000.1733.

3 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On Compiling CNFs into Structured Deterministic DNNFs. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference*, volume 9340 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2015.

4 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge Compilation Meets Communication Complexity. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1008–1014. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/147.

5 Simone Bova and Stefan Szeider. Circuit Treewidth, Sentential Decision, and Query Compilation. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 233–246. ACM, 2017. doi:10.1145/3034786.3034787.

6 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding Model Counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science*, pages 143–156, 2015.

7 Florent Capelli and Stefan Mengel. Knowledge Compilation, Width and Quantification. *CoRR*, abs/1807.04263, 2018. arXiv:1807.04263.

8 Hubie Chen. Quantified Constraint Satisfaction and Bounded Treewidth. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 161–165, 2004.

9 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.

10 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 819–826. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-143.

11 Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.

12 Wolfgang Dvorák, Stefan Szeider, and Stefan Woltran. Reasoning in Argumentation Frameworks of Bounded Clique-Width. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo Ricardo Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010.*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 219–230. IOS Press, 2010. doi:10.3233/978-1-60750-619-5-219.

13 Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. Treewidth in Verification: Local vs. Global. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*, pages 489–503. Springer, 2005. doi:10.1007/11591191_34.

14 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Exploiting Treewidth for Projected Model Counting and Its Limits. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 165–184. Springer, 2018. doi:10.1007/978-3-319-94144-8_11.

15 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. doi:10.1016/j.dam.2006.06.020.

16 Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010. doi:10.1016/j.artint.2009.10.003.

17 Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to courcelle's theorem. *CoRR*, abs/1805.08456, 2018. accepted for SAT'18. arXiv:1805.08456.

18 Michael Lampis and Valia Mitsou. Treewidth with a Quantifier Alternation Revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

19 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. *Algorithmica*, 76(1):168–194, 2016.

20 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. *Algorithmica*, 76(1):168–194, 2016. doi:10.1007/s00453-015-0030-x.

21 Knot Pipatsrisawat and Adnan Darwiche. New Compilation Languages Based on Structured Decomposability. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 517–522. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-082.php.

22 Thammanit Pipatsrisawat and Adnan Darwiche. A Lower Bound on the Size of Decomposable Negation Normal Form. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1856.

23 S. Hortemo Sæther, J.A. Telle, and M. Vatshelle. Solving MaxSAT and #SAT on Structured CNF Formulas. In *Theory and Applications of Satisfiability Testing*, pages 16–31, 2014.

24 M. Samer and S. Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50–64, 2010.

25 Martin Sauerhoff. Approximation of boolean functions by combinatorial rectangles. *Theor. Comput. Sci.*, 1-3(301):45–78, 2003. doi:10.1016/S0304-3975(02)00568-6.

26 Friedrich Slivovsky and Stefan Szeider. Model Counting for Formulas of Bounded Clique-Width. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 677–687. Springer, 2013. doi:10.1007/978-3-642-45030-3_63.

27 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability, 6th International Conference*, volume 2919 of *LNCS*, pages 188–202. Springer, 2004.