Connecting Width and Structure in Knowledge Compilation

Antoine Amarilli[1], Mikaël Monet[1,3], and Pierre Senellart[1,2,3]

1 **LTCI, Télécom ParisTech, Université Paris-Saclay**
2 **DI ENS, ENS, CNRS, PSL Research University; Paris, France**
3 **Inria Paris; Paris, France**

Abstract

Several query evaluation tasks can be done via *knowledge compilation*: the query result is compiled as a *lineage circuit* from which the answer can be determined. For such tasks, it is important to leverage some width parameters of the circuit, such as bounded treewidth or pathwidth, to convert the circuit to structured classes, e.g., deterministic structured NNFs (d-SDNNFs) or OBDDs. In this work, we show how to connect the width of circuits to the size of their structured representation, through upper and lower bounds. For the upper bound, we show how bounded-treewidth circuits can be converted to a d-SDNNF, in time linear in the circuit size. Our bound, unlike existing results, is constructive and only singly exponential in the treewidth. We show a related lower bound on monotone DNF or CNF formulas, assuming a constant bound on the arity (size of clauses) and degree (number of occurrences of each variable). Specifically, any d-SDNNF (resp., SDNNF) for such a DNF (resp., CNF) must be of exponential size in its treewidth; and the same holds for pathwidth when compiling to OBDDs. Our lower bounds, in contrast with most previous work, apply to *any* formula of this class, not just a well-chosen family. Hence, for our language of DNF and CNF, pathwidth and treewidth respectively characterize the efficiency of compiling to OBDDs and (d-)SDNNFs, that is, compilation is singly exponential in the width parameter. We conclude by applying our lower bound results to the task of query evaluation.

1998 ACM Subject Classification H.2 DATABASE MANAGEMENT

Keywords and phrases knowledge compilation; probabilistic databases; treewidth; circuits

Digital Object Identifier 10.4230/LIPIcs.ICDT.2018.6

1 Introduction

Uncertainty and errors in data can be modeled using *probabilistic databases* [39], annotating every tuple with a probability of existence. Query evaluation on probabilistic databases must then handle the uncertainty by computing the probability that each query result holds. A common technique to evaluate queries on probabilistic databases is the *intensional approach*: first compute a representation of the *lineage* of the query on the database, which intuitively describes how the query depends on the possible database facts; then use this lineage to compute probabilities efficiently. Specifically, the lineage can be computed as a *circuit* [32], and efficient probability computation can be achieved by restricting to tractable circuit classes via *knowledge compilation*. Thus, to evaluate queries on probabilistic databases, we can use knowledge compilation algorithms to translate circuits to tractable classes; conversely, lower bounds in knowledge compilation can identify the limits of the intensional approach.

In this paper, we study the relationship between two kinds of tractable circuit classes in knowledge compilation: *width-based* classes, specifically, bounded-treewidth and bounded-pathwidth circuits; and *structure-based* classes, specifically, OBDDs (ordered binary decision

21st International Conference on Database Theory (ICDT 2018).
Editors: Benny Kimelfeld and Yael Amsterdamer; Article No. 6; pp. 6:1–6:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

diagrams [17], following a *variable order*) and d-SDNNFs (structured deterministic decomposable negation normal forms [35], following a *v-tree*). Circuits of bounded treewidth can be obtained as a result of practical query evaluation [3, 6, 30], whereas OBDDs and d-DNNFs have been studied to show theoretical characterizations of the query lineages they can represent [31]. Both classes enjoy tractable probabilistic computation: for width-based classes, using *message passing* [33], in time linear in the circuit and exponential in the treewidth; for OBDDs and d-SDNNFs, in linear time by definition of the class [22]. Hence the question that we study: can we compile width-based classes efficiently into structure-based classes?

We first study how to perform this transformation, and show corresponding *upper bounds*. Existing work has already studied the compilation of bounded-pathwidth circuits to OBDDs [32], which can be made constructive [7, Lemma 6.9]. Accordingly, we focus on compiling *bounded-treewidth circuits* to *d-SDNNF circuits*. Our first contribution, stated in Section 3 and proved in Section 4, is to show the following:

▶ **Result 1** (Theorem 5 and subsequent remark). *Given as input a Boolean circuit C of treewidth k, we can compute a d-SDNNF equivalent to C in time $O(|C| \times f(k))$ where f is singly exponential.*

The algorithm transforms the input circuit bottom-up, considering all possible valuations of the gates in each bag of the tree decomposition, and keeping track of additional information to remember which guessed values have been substantiated by a corresponding input. Our result relates to a recent theorem of Bova and Szeider in [16], except that our bound depends on $|C|$ (the circuit size) whereas their bound depends on the number of variables of C. In exchange for this, we improve on their result in two ways. First, our result is constructive, whereas [16] only shows a bound on the size of the d-SDNNF, without bounding the complexity of effectively computing it. Second, our bound is singly exponential in k, whereas [16] is doubly exponential; this allows us to be competitive with message passing (also singly exponential in k), and we believe it can be useful for practical applications. Indeed, beyond probabilistic query evaluation, our result implies that all tractable tasks on d-SDNNFs (e.g., enumeration [2] and MAP inference [27]) are also tractable on bounded-treewidth circuits.

Second, we study *lower bounds* on how efficiently we can convert from width-based to structure-based classes. Our bounds already apply to a weaker formalism of width-based circuits, namely monotone CNFs and DNFs of bounded width, so we focus on them. Our second contribution (in Section 5) concerns pathwidth and OBDD representations: we show that, up to factors in the formula arity (maximal size of clauses) and degree (maximal number of variable occurrences), any OBDD for a monotone CNF or DNF must be of width exponential in the pathwidth of the formula. Formally:

▶ **Result 2** (Theorem 15). *Let φ be a monotone DNF or monotone CNF, let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any OBDD for φ has width $2^{\Omega\left(\frac{\mathrm{pw}(\varphi)}{a^3 \times d^2}\right)}$.*

This result generalizes several existing lower bounds in knowledge compilation that exponentially separate CNFs from OBDDs, such as [25] and [15, Theorem 19].

Our third contribution (Section 6) is to show an analogue for treewidth and (d-)SDNNFs:

▶ **Result 3** (Theorem 25). *Let φ be a monotone DNF (resp., monotone CNF), let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any d-SDNNF (resp., SDNNF) for φ has size $2^{\Omega\left(\frac{\mathrm{tw}(\varphi)}{a^3 \times d^2}\right)}$.*

Our two lower bounds contribute to a vast landscape of knowledge compilation results giving lower bounds on compiling specific Boolean functions to restricted circuits classes, e.g., [15, 25, 37] to OBDDs, [18] to *decision* structured DNNF, [9] to *sentential decision diagrams* (SDDs), [13, 36] to d-SDNNF, [13, 19, 20] to d-DNNFs and DNNFs. However, all those lower bounds (with the exception of some results in [19, 20] discussed in Section 6)

apply to well-chosen families of Boolean functions (usually CNF), whereas Result 2 and 3 apply to *any* monotone CNF and DNF. Together with Result 1, these generic lower bounds point to a strong relationship between width parameters and structure representations, on monotone CNFs and DNFs of constant arity and degree. Specifically, the smallest width of OBDD representations of any such formula φ is in $2^{\Theta(\mathrm{pw}(\varphi))}$, i.e., precisely singly exponential in the pathwidth; and an analogous bound applies to d-SDNNF size and treewidth of DNFs.

To prove our lower bounds, we rephrase pathwidth and treewidth to new notions of *pathsplitwidth* and *treesplitwidth*, which intuitively measure the performance of a variable ordering or v-tree. We also use the *disjoint non-covering prime implicant sets* (dncpi-sets), a tool introduced in [1, 7] by some of the present authors, and generalizing *subfunction width* [15]. These dncpi-sets allow us to derive lower bounds on OBDD width directly using [1]. We show how they can also imply lower bounds on d-SDNNF size, using the recent communication complexity approach of Bova, Capelli, Mengel and Slivovsky [13].

Our fourth contribution (Section 7) applies our lower bounds to intensional query evaluation on relational databases. We reuse the notion of *intricate* queries of [7], and show that d-SDNNF representations of the lineage of these queries have size exponential in the treewidth of *any* input instance. This extends the result of [7] from OBDDs to d-SDNNFs:

▶ **Result 4** (Theorem 33). *There is a constant $d \in \mathbb{N}$ such that the following is true. Let σ be an arity-2 signature, and Q be a connected* $\mathrm{UCQ}^{\neq}$ *which is intricate on σ. For any instance I on σ, any d-SDNNF representing the lineage of Q on I has size $\geqslant 2^{\Omega(\mathrm{tw}(I)^{1/d})}$.*

As in [7], this result shows that, on arity-2 signatures and under constructibility assumptions, treewidth is the right parameter on instance families to ensure that all queries (in monadic second-order) have tractable d-SDNNF lineage representations.

We start in Section 2 with preliminaries. Full proofs of all results are provided in the extended version [8].

2 Preliminaries

Hypergraphs, treewidth, pathwidth. A *hypergraph* $H = (V, E)$ consists of a finite set of *nodes* (or *vertices*) V and of a set E of *hyperedges* (or simply *edges*) which are non-empty subsets of V. We always assume that hypergraphs have at least one edge. For a node v of H, we write $E(v)$ for the set of edges of H that contain v. The *arity* of H, written $\mathrm{arity}(H)$, is the maximal size of an edge of H. The *degree* of H, written $\mathrm{degree}(H)$, is the maximal number of edges to which a vertex belongs, i.e., $\max_{v \in V} |E(v)|$.

A *tree decomposition* of a hypergraph $H = (V, E)$ is a finite, rooted tree T, whose nodes b (called *bags*) are labeled by a subset $\lambda(b)$ of V, and which satisfies:

1. for every fact $e \in E$, there is a bag $b \in T$ with $e \subseteq \lambda(b)$;
2. for all $v \in V$, the set of bags $\{b \in T \mid v \in \lambda(b)\}$ is a connected subtree of T.

For brevity, we identify a bag b with its domain $\lambda(b)$. The *width* of T is $\max_{b \in T} |\lambda(b)| - 1$. The *treewidth* of H is the minimal width of a tree decomposition of H. Pathwidth is defined similarly but with *path decompositions*, where T is a path rather than a tree.

It is NP-hard to determine the treewidth of a hypergraph, but we can compute a tree decomposition in linear time when parametrizing by the treewidth:

▶ **Theorem 1** ([10]). *Given a hypergraph H and an integer $k \in \mathbb{N}$ we can check in time $O(|H| \times g(k))$ whether H has treewidth $\leqslant k$, and if yes output a tree decomposition of H of width $\leqslant k$, where g is a fixed function in $O(2^{(32+\varepsilon)k^3})$ for any $\varepsilon > 0$.*

For simplicity, we will often assume that a tree decomposition is *nice*, meaning that:

1. it is a full binary tree, i.e., each node has exactly zero or two children;
2. for every internal bag b with children b_l, b_r we have $b \subseteq b_l \cup b_r$;
3. for every leaf bag b we have $|b| \leqslant 1$;
4. for every non-root bag b with parent b', we have $|b \setminus b'| \leqslant 1$;
5. for the root bag b we have $|b| \leqslant 1$.

▶ **Lemma 2.** *Given a tree decomposition T of width k having n nodes, we can compute in time $O(k \times n)$ a nice tree decomposition T' of width k having $O(k \times n)$ nodes.*

Boolean circuits and functions. A (Boolean) *valuation* of a set V is a function $\nu : V \to \{0,1\}$. A *Boolean function* φ on variables V is a mapping that associates to each valuation ν of V a Boolean value in $\{0,1\}$ called the *evaluation* of φ according to ν.

A *(Boolean) circuit* $C = (G, W, g_{\mathsf{output}}, \mu)$ is a directed acyclic graph (G, W) whose vertices G are called *gates*, whose edges W are called *wires*, where $g_{\mathsf{output}} \in G$ is the *output gate*, and where each gate $g \in G$ has a *type* $\mu(g)$ among var (a *variable gate*), not, or, and. The *inputs* of a gate $g \in G$ are the gates $g' \in G$ such that $(g', g) \in W$; the *fan-in* of g is its number of inputs. We require not-gates to have fan-in 1 and var-gates to have fan-in 0. The *treewidth* of C, and its *size*, are those of the graph (G, W). The set C_{var} of *variable gates* of C are those of type var. Given a valuation ν of C_{var}, we extend it to an *evaluation* of C by mapping each variable $g \in C_{\mathsf{var}}$ to $\nu(g)$, and evaluating the other gates according to their type. The Boolean function on C_{var} *captured* by the circuit is the one that maps ν to the evaluation of g_{output} under ν. Two circuits are *equivalent* if they capture the same function.

We recall restricted circuit classes from knowledge compilation. We say that C is in *negation normal form* (NNF) if the inputs of not-gates are always variable gates. For a gate g in a Boolean circuit C, we write $\mathsf{Vars}(g)$ for the set of variable gates of C_{var} that have a directed path to g in C. An and-gate g of C is *decomposable* if for every two input gates $g_1 \neq g_2$ of g we have $\mathsf{Vars}(g_1) \cap \mathsf{Vars}(g_2) = \emptyset$. We call C *decomposable* if each and-gate is.

A stronger requirement than decomposability is *structuredness*. A *v-tree* [35] over a set V is a rooted ordered binary tree T whose leaves are in bijection with V; we identify each leaf with the associated element of V. For $n \in T$, we denote by T_n the subtree of T rooted at n, and for a subset $U \subseteq T$ of nodes of T, we denote by $\mathsf{Leaves}(U)$ the leaves that are in U, i.e., $U \cap V$. We say that T *structures* a Boolean circuit C (and call it a *v-tree for* C) if T is over the set C_{var} and if, for every and-gate g of C with inputs $g_1, \ldots, g_m$ and $m > 0$, there is a node $n \in T$ that *structures* g, i.e., n has m children $n_1, \ldots, n_m$ and we have $\mathsf{Vars}(g_i) \subseteq \mathsf{Leaves}(T_{n_i})$ for all $1 \leqslant i \leqslant m$. We call C *structured* if some v-tree structures it. Note that structured Boolean circuits are always decomposable, and their and-gates have at most two inputs because T is binary.

A last requirement on circuits is *determinism*. An or-gate g of C is *deterministic* if there is no pair $g_1 \neq g_2$ of input gates of g and valuation ν of C_{var} such that g_1 and g_2 both evaluate to 1 under ν. A Boolean circuit is *deterministic* if each or-gate is.

The main structured class of circuits that we study in this work are *deterministic structured decomposable NNFs*, which we denote d-SDNNF for brevity as in [35].

DNFs and CNFs. We also study other representations of Boolean functions, namely, Boolean formulas in *conjunctive normal form* (*CNFs*) and in *disjunctive normal form* (*DNFs*). A DNF (resp., CNF) φ on a set of variables V is a disjunction (resp., conjunction) of *clauses*, each of which is a conjunction (resp., disjunction) of *literals* on V, i.e., variables of V (a *positive* literal) or their negation (a *negative* literal). A *monotone DNF* (resp., monotone CNF) is one where all literals are positive, in which case we often identify a clause to the

set of variables that it contains. We always assume that monotone DNFs and monotone CNFs are *minimized*, i.e., no clause is a subset of another. This ensures that every monotone Boolean function has a unique representation as a monotone DNF (the disjunction of its prime implicants), and likewise for CNF. We assume that CNFs and DNFs always contain at least one non-empty clause (in particular, they cannot represent constant functions). Monotone DNFs and CNFs φ are isomorphic to hypergraphs: the vertices are the variables of φ, and the hyperedges are the clauses of φ. We often identify φ to its hypergraph. In particular, the *pathwidth* and *treewidth* of φ, and its *arity* and *degree*, are defined as that of its hypergraph.

3 Upper Bounds

Our upper bound result studies how to compile a Boolean circuit to a d-SDNNF, parametrized by the treewidth of the input circuit. To present it, we first review the independent result that was recently shown by Bova and Szeider [16] about these circuit classes:

▶ **Theorem 3** ([16, Theorem 3 and Equation (22)]). *Given a Boolean circuit C with n variables and of treewidth $\leqslant k$, there exists an equivalent d-SDNNF of size $O(f(k) \times n)$, where f is doubly exponential.*

An advantage of their result is that it depends only on the *number of variables* of the circuit (and on the width parameter), not on the *size* of the circuit. None of our results will have this advantage, and we will always measure complexity as a function of the size of the input circuit. In exchange for this advantage, their result has two drawbacks: (i) the doubly exponential dependency on the width; and (ii) its nonconstructive aspect, because [16] gives no time bound on the computation, leaving open the question of effectively compiling bounded-treewidth circuits to d-SDNNFs.

Naive constructive bound. We first address the second drawback by showing an easy constructive result. The argument is very simple and appeals to techniques from our earlier works on provenance circuits [6, 7]; it is independent from the techniques of [16].

▶ **Theorem 4.** *Given any circuit C of treewidth k, we can compute an equivalent d-SDNNF in linear time parametrized by k, i.e., in time $O(|C| \times f(k))$ for some computable function f.*

Proof sketch. We encode in linear time the input circuit C to a relational instance I with same treewidth. We use [7, Theorem 6.11] to construct in linear time a provenance representation C' on I of a fixed MSO formula that describes Boolean circuit evaluation. This allows us to obtain in linear time from C' the desired equivalent d-SDNNF representation. ◀

This result shows that we can effectively compile in linear time parametrized by the treewidth k, but does not address the first drawback, namely, the dependency in k.

Improved bound. Our main upper bound result subsumes the naive bound above, with a more elaborate proof, again independent of the techniques of [16]. It addresses both drawbacks and shows that we can effectively compile in time singly exponential in k; formally:

▶ **Theorem 5.** *Given as input a Boolean circuit C and tree decomposition T of width k, we can compute a d-SDNNF equivalent to C with its v-tree, in $O\left(|T| \times 2^{(4+\varepsilon)k}\right)$ for any $\varepsilon > 0$.*

We prove Theorem 5 in the next section. Observe how we assume the tree decomposition to be given as part of the input. If it is not, we can compute one with Theorem 1, but this becomes the bottleneck: the complexity becomes $O\left(|C| \times 2^{(32+\varepsilon)k^3}\right)$ for any $\varepsilon > 0$.

Applications. Theorem 5 implies several consequences for bounded-treewidth circuits. The first one deals with *probability computation*: we are given a *probability valuation* π mapping each variable $g \in C_{\mathsf{var}}$ to a probability that g is true (independently from other variables), and we wish to compute the probability $\pi(C)$ that C evaluates to true under π, assuming that arithmetic operations (sum and product) take unit time. This problem is #P-hard for arbitrary circuits, but it is tractable for d-SDNNF [22]. Hence, our result implies the following, where $|\pi|$ denotes the size of writing the probability valuation π:

▶ **Corollary 6.** *Given a Boolean circuit C, a tree decomposition T of width k of C, and a probability valuation π of C, we can compute $\pi(C)$ in $O\left(|\pi| + |T| \times 2^{(4+\varepsilon)k}\right)$ for any $\varepsilon > 0$.*

This improves the bound obtained when applying message passing techniques [33] directly on the bounded-treewidth input circuit (as presented, e.g., in [5, Theorem D.2]). Indeed, message passing applies to *moralized* representations of the input: for each gate, the tree decomposition must contain a bag containing all inputs of this gate *simultaneously*, which is problematic for circuits of large fan-in. Indeed, if the original circuit has a tree decomposition of width k, rewriting it to make it moralized results in a tree decomposition of width $3k^2$ (see [4, Lemmas 53 and 55]), and the bound of [5, Theorem D.2] then yields an overall complexity of $O\left(|\pi| + |T| \times 2^{3k^2}\right)$ for message passing. Our Corollary 6 achieves a more favorable bound because Theorem 5 uses directly the associativity of and and or. We note that the connection between message-passing techniques and structured circuits has also been investigated by Darwiche, but his result [23, Theorem 6] produces arithmetic circuits rather than d-DNNFs, and it also needs the input to be moralized.

A second consequence concerns the task of *enumerating* the accepting valuations of circuits, i.e., producing them one after the other, with small *delay* between each accepting valuation. The valuations are concisely represented as *assignments*, i.e., as a set of variables that are set to true, omitting those that are set to false. This task is of course NP-hard on arbitrary circuits (as it implies that we can check whether an accepting valuation exists), but was recently shown in [2] to be feasible on d-SDNNFs with linear-time preprocessing and delay linear in the Hamming weight of each produced assignment. Hence, we have:

▶ **Corollary 7.** *Given a Boolean circuit C and a tree decomposition T of width k of C, we can enumerate the accepting assignments of C with preprocessing in $O\left(|T| \times 2^{(4+\varepsilon)k}\right)$ and delay linear in the size of each produced assignment.*

Other applications of Theorem 5 include counting the number of satisfying valuations of the circuit (a special case of probability computation), MAP inference [27] or random sampling of possible worlds (which can be done on the d-SDNNF in an easy manner).

4 Proof of the Main Upper Bound Result

In this section, we present the construction used to prove Theorem 5. We start with prerequisites, and then describe how to build the d-SDNNF equivalent to the input bounded-treewidth circuit. Last, we sketch the correctness proof.

Prerequisites. Let C be the input circuit, and T the input tree decomposition. By Lemma 2, we assume that T is nice. Further, up to adding a constant number of bags and re-rooting T, we can assume that the root bag of T contains only the output gate g_{output}. For any bag b of T, we define $\mathrm{VarT}(b)$ to be the set of variable gates such that b is the topmost bag in which they appear; as T is nice, $\mathrm{VarT}(b)$ is either empty or is a singleton $\{g\}$, in which case we call b *responsible for the variable gate* g. We can explicitly compute the function VarT in $O(|T|)$, i.e., compute $\mathrm{VarT}(b)$ for each $b \in T$; see for instance [28, Lemma 3.1].

To abstract away the type of gates and their values in the construction, we will talk of *strong* and *weak* values. Intuitively, a value is *strong* for a gate g if any input g' of g which carries this value determines the value of g; and *weak* otherwise. Formally:

▶ **Definition 8.** Let g be a gate and $c \in \{0,1\}$:

- If g is an and-gate, we say that $c = 0$ is *strong* for g and $c = 1$ is *weak* for g;
- If g is an or-gate, we say that $c = 1$ is *strong* for g and $c = 0$ is *weak* for g;
- If g is a not-gate, $c = 0$ and $c = 1$ are both *strong* for g;
- For technical convenience, if g is a var-gate, $c = 0$ and $c = 1$ are both *weak* for g.

If we take any valuation $\nu : C_{\mathsf{var}} \to \{0,1\}$ of the circuit $C = (G, W, g_{\mathsf{output}}, \mu)$, and extend it to an evaluation $\nu : G \to \{0,1\}$, then ν will respect the semantics of gates. In particular, it will *respect strong values*: for any gate g of C, if g has an input g' for which $\nu(g')$ is a strong value, then $\nu(g)$ is determined by $\nu(g')$, specifically, it is $\nu(g')$ if g is an or- or an and-gate, and $1 - \nu(g')$ if g is a not-gate. In our construction, we will need to guess how gates of the circuit are evaluated, focusing on a subset of the gates (as given by a bag of T); we will then call *almost-evaluation* an assignment of these gates that respects strong values. Formally:

▶ **Definition 9.** Let U be a set of gates of C. We call $\nu : U \to \{0,1\}$ a (C,U)-*almost-evaluation* if it *respects strong values*, i.e., for every gate $g \in U$, if there is an input g' of g in U and $\nu(g')$ is a strong value for g, then $\nu(g)$ is determined from $\nu(g')$ in the sense above.

Respecting strong values is a necessary condition for such an assignment to be extensible to a valuation of the entire circuit. However, it is not sufficient: an almost-evaluation ν may map a gate g to a strong value even though g has no input that can justify this value. This is hard to avoid: when we focus on the set U, we do not know about other inputs of g. For now, let us call *unjustified* the gates of U that carry a strong value that is not justified by ν:

▶ **Definition 10.** Let U be a set of gates of a circuit C and ν a (C,U)-almost-evaluation. We call $g \in U$ *unjustified* if $\nu(g)$ is a strong value for g, but, for every input g' of g in U, the value $\nu(g')$ is weak for g; otherwise, g is *justified*. The set of unjustified gates is written $\mathrm{Unj}(\nu)$.

Let us start to explain how to construct the d-SDNNF circuit D equivalent to the input circuit C. We do so by traversing T bottom-up, and for each bag b of T we create gates $G_b^{\nu,S}$ in D, where ν is a (C,b)-almost-evaluation and S is a subset of $\mathrm{Unj}(\nu)$ which we call the *suspicious gates* of $G_b^{\nu,s}$. We will connect the gates of D created for each internal bag b with the gates created for its children in T, in a way that we will explain later. Intuitively, for a gate $G_b^{\nu,S}$ of D, the *suspicious gates* g in the set S are gates of b whose strong value is not justified by ν (i.e., $g \in \mathrm{Unj}(\nu)$), and is not justified either by any of the almost-evaluations at descendant bags of b to which $G_b^{\nu,S}$ is connected. We call *innocent* the other gates of b; they are the gates that are justified in ν (in particular, those who carry weak values), and the gates that are unjustified in ν but have been justified by an almost-evaluation at a descendant bag b' of b. Crucially, in the latter case, the gate g' justifying the strong value in b' may no longer appear in b, making g unjustified for ν; this is why we remember the set S.

We still have to explain how we connect the gates $G_b^{\nu,S}$ of D to the gates $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ created for the children b_l and b_r of b in T. The first condition is that ν_l and ν_r must *mutually agree*, i.e., $\nu_l(g) = \nu_r(g)$ for all $g \in b_l \cap b_r$, and ν must then be the union of ν_l and ν_r, restricted to b. Remember that T is nice, so b is a subset of $b_l \cup b_r$, and it is easy to verify that ν is then a (C,b)-almost-evaluation. We impose a second condition to prohibit suspicious gates from escaping before they have been justified, which we formalize as *connectibility* of a pair (ν, S) at bag b to the parent bag of b.

▶ **Definition 11.** Let b be a non-root bag, b' its parent bag, and ν a (C,b)-almost-evaluation. For any set $S \subseteq \mathrm{Unj}(\nu)$, we say that (ν, S) is *connectible* to b' if $S \subseteq b'$, i.e., the suspicious gates of ν must still appear in b'.

If a gate $G_b^{\nu,S}$ is such that (ν, S) is not connectible to the parent bag b', then this gate will not be used as input to any other gate (but we do not try to preemptively remove these useless gates in the construction). We are now ready to give the formal definition that will be used to explain how gates are connected:

▶ **Definition 12.** Let b be an internal bag with children b_l and b_r, let ν_l and ν_r be respectively (C, b_l) and (C, b_r)-almost-evaluations that mutually agree, and $S_l \subseteq \mathrm{Unj}(\nu_l)$ and $S_r \subseteq \mathrm{Unj}(\nu_r)$ be sets of suspicious gates such that both (ν_l, S_l) and (ν_r, S_r) are connectible to b. The *result* of (ν_l, S_l) and (ν_r, S_r) is then defined as the pair (ν, S) where:

- ν is a (C, b)-almost-evaluation defined as the restriction of $\nu_l \cup \nu_r$ to b.
- $S \subseteq \mathrm{Unj}(\nu)$ is the new set of suspicious gates, defined as follows. A gate $g \in b$ is innocent (i.e., $g \in b \setminus S$) if it is justified for ν or if it is innocent for some child. Formally, $b \setminus S := (b \setminus \mathrm{Unj}(\nu)) \cup \big[b \cap [(b_l \setminus S_l) \cup (b_r \setminus S_r)]\big]$.

Construction. We now use these definitions to present the construction formally. For every variable gate g of C, we create a corresponding variable gate $G^{g,1}$ of D, and we create $G^{g,0} := \mathsf{not}(G^{g,1})$. For every internal bag b of T, for each (C, b)-almost-evaluation ν and set $S \subseteq \mathrm{Unj}(\nu)$ of suspicious gates of ν, we create one and-gate $G_b^{\nu,S}$ and one or-gate $G_{b,\mathsf{children}}^{\nu,S}$ which is an input of $G_b^{\nu,S}$. For every leaf bag b of T, we create one gate $G_b^{\nu,S}$ for every (C, b)-almost-evaluation ν, where we set $S := \mathrm{Unj}(\nu)$; intuitively, in a leaf bag, an unjustified gate is always suspicious (it cannot have been justified at a descendant bag).

Now, for each internal bag b of T with children b_l, b_r, for each pair of gates $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ that are both connectible to b and where ν_l and ν_r mutually agree, letting (ν, S) be the result of (ν_l, S_l) and (ν_r, S_r), we create a gate $G_b^{\nu_l,S_l,\nu_r,S_r} = \mathsf{and}(G_{b_l}^{\nu_l,S_l}, G_{b_r}^{\nu_r,S_r})$ and make it an input of $G_{b,\mathsf{children}}^{\nu,S}$. Last, for each bag b which is responsible for a variable gate g (i.e., $\mathrm{VarT}(b) = \{g\}$), for each (C, b)-almost-evaluation ν and set of suspicious gates $S \subseteq \mathrm{Unj}(\nu)$, we set the gate $G^{g,\nu(g)}$ to be the second input of $G_b^{\nu,S}$. The output gate of D is the gate $G_{b_{\mathrm{root}}}^{\nu,\emptyset}$ where b_{root} is the root of T and ν maps g_{output} to 1 (remember that b_{root} contains only g_{output}).

Correctness. We have formally described the construction of our d-SDNNF D. The construction clearly works in linear time, and we can prove that the dependency on k of the running time is as stated. Further, we easily see that D is structured by a v-tree constructed from the tree decomposition T. To show that D is equivalent to C, one direction is easier: any valuation χ that satisfies C also satisfies D, because we can construct an *accepting trace* in D using the gates $G_b^{\nu,S}$ for ν the restriction of the evaluation χ to b, and for $S := \mathrm{Unj}(\chi_{|T_b})$ where T_b denotes the gates of C occurring in the bags of the subtree of T rooted at b. The converse is trickier: we show that any accepting trace of D describes an evaluation of C that respects strong values by definition of almost-evaluations, and eventually justifies every gate which is given a strong value thanks to our bookkeeping of suspicious gates. Last, we show that D is deterministic: this is unexpected because we freely guess the values of gates of C at leaf bags, but it holds because, when we know the valuation of the variable gates, knowing the valuation of all gates of a bag b uniquely fixes the valuation at the subtree rooted at b. This concludes the proof sketch of Theorem 5; see the extended version [8] for the full proof.

5 Lower Bounds on OBDDs

We now move to lower bounds on the size of structured representations of Boolean functions, in terms of the width of a circuit for that function. Our end goal is to obtain a lower bound for (d-)SDNNFs, that will form a counterpart to the upper bound of Theorem 5. We will do

so in Section 6. For now, in this section, we consider a weaker class of lineage representations than (d-)SDNNFs, namely, *OBDDs*.

▶ **Definition 13.** An *ordered binary decision diagram* (or *OBDD*) on a set of variables $V = \{v_1, \ldots, v_n\}$ is a rooted DAG O whose leaves are labeled by 0 or 1, and whose internal nodes are labeled with a variable of V and have two outgoing edges labeled 0 and 1. We require that there exists a total order $\mathbf{v} = v_{i_1}, \ldots, v_{i_n}$ on the variables such that, for every path from the root to a leaf, the sequence of variables which labels the internal nodes of the path is a subsequence of $\mathbf{v}$ and does not contain duplicate variables. The OBDD O *captures* a Boolean function on V defined by mapping each valuation ν to the value of the leaf reached from the root by following the path given by ν. The *size* $|O|$ of O is its number of nodes, and the *width* w of O is the maximum number of nodes at every *level*, where a level is defined for a prefix of $\mathbf{v}$ as the set of nodes reached by enumerating all possible valuations of this prefix. Note that we clearly have $|O| \leqslant |V| \times w$.

Our upper bound in the previous section applied to arbitrary Boolean circuits; however, our lower bounds in this section and the next one will already apply to much weaker formalisms for Boolean functions, namely, monotone DNFs and monotone CNFs (recall their definition from Section 2). Some lower bounds are already known for the compilation of monotone CNFs into OBDDs: Bova and Slivovsky have constructed a family of CNFs of bounded degree whose OBDD width is exponential in their number of variable occurrences [15, Theorem 19], following an earlier result of this type by Razgon [37, Corollary 1]. The result is as follows:

▶ **Theorem 14** ([15, Theorem 19])**.** *There is a class of monotone CNF formulas of bounded degree and arity such that every formula φ in this class has OBDD size at least $2^{\Omega(|\varphi|)}$.*

We adapt some of these techniques to show a more general result: our lower bound applies to *any* monotone DNF or monotone CNF, not to one specific family. Specifically, we show:

▶ **Theorem 15.** *Let φ be a monotone DNF or monotone CNF, let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any OBDD for φ has width $\geqslant 2^{\left\lfloor \frac{\mathrm{pw}(\varphi)}{a^3 \times d^2} \right\rfloor}$.*

From our Theorem 15, we can easily derive Theorem 14 using the fact (also used in the proof of [15, Theorem 19]) that there exists a family of monotone CNFs of bounded degree and arity whose treewidth (hence pathwidth) is linear in their size, namely, the CNFs built from *expander graphs* (see [29, Theorem 5 and Proposition 1]). Note that expander graphs can also be used to show lower bounds for *(non-deterministic and non-structured) DNNFs* for a CNF formula [12]; our lower bound on SDNNFs of Section 6 does not capture this result (because we need structuredness).

We observe that, for a family of formulas with bounded arity and degree, the bound of Theorem 15 is optimal, up to constant factors in the exponent. Indeed, following earlier work [26, 37], Bova and Slivovsky have shown that any CNF φ can be compiled to OBDDs of width $2^{\mathrm{pw}(\varphi)+2}$ [15, Theorem 4 and Lemma 9]. (Their upper bound result also applies to DNFs, and does not assume monotonicity nor a bound on the arity or degree.) In other words, for any monotone DNF or monotone CNF of bounded arity and degree, pathwidth *characterizes* the width of an OBDD for the formula, in the following sense:

▶ **Corollary 16.** *For any constant c, for any monotone DNF (or monotone CNF) φ with arity and degree bounded by c, the smallest width of an OBDD for φ is $2^{\Theta(\mathrm{pw}(\varphi))}$.*

This corollary talks about the pathwidth of φ measured as that of its hypergraph, but note that the same result would hold when measuring the pathwidth of the incidence graph or dual hypergraph of φ. Indeed, all these pathwidths are within a constant factor of one another when the degree and arity are bounded by a constant.

We prove Theorem 15 in the rest of this section. We present the proof in the case of monotone DNFs to reuse existing lower bound techniques from [1,7], but explain at the end of this section how the proof adapts to monotone CNFs. We first present *pathsplitwidth*, a new notion which intuitively measures the performance of a variable ordering for an OBDD on the monotone DNF φ, and connect it to the pathwidth of φ. Second, we recall the definition of *dncpi-sets* introduced in [1,7] to show lower bounds from the structure of Boolean functions. Last, we conclude the proof by connecting pathsplitwidth to the size of dncpi-sets.

Pathsplitwidth. The first step of the proof is to rephrase the bound on pathwidth, arity, and degree, in terms of a bound on the performance of variable orderings. Intuitively, a good variable ordering is one which does not *split* too many clauses. Formally:

▶ **Definition 17.** Let $H = (V, E)$ be a hypergraph, and $\mathbf{v} = v_1, \ldots, v_{|V|}$ be an ordering on the variables of V. For $1 \leqslant i \leqslant |V|$, we define $\mathrm{Split}_i(\mathbf{v}, H)$ as the set of hyperedges e of H that contain both a variable at or before v_i, and a variable strictly after v_i, formally: $\mathrm{Split}_i(\mathbf{v}, H) := \{e \in E \mid \exists l \in \{1, \ldots, i\} \text{ and } \exists r \in \{i+1, \ldots, |V|\} \text{ such that } \{v_l, v_r\} \subseteq e\}$. Note that $\mathrm{Split}_{|V|}(\mathbf{v}, H)$ is always empty. The *pathsplitwidth* of $\mathbf{v}$ relative to H is the maximum size of the split, formally, $\mathrm{psw}(\mathbf{v}, H) := \max_{1 \leqslant i \leqslant |V|} |\mathrm{Split}_i(\mathbf{v}, H)|$. The *pathsplitwidth* $\mathrm{psw}(H)$ of H is then the minimum of $\mathrm{psw}(\mathbf{v}, H)$ over all variable orderings $\mathbf{v}$ of V.

In other words, $\mathrm{psw}(H)$ is the smallest integer $n \in \mathbb{N}$ such that, for any variable ordering $\mathbf{v}$ of the nodes of H, there is a moment at which n hyperedges of H are *split*, i.e., for n hyperedges e, we have begun enumerating the nodes of e but we have not enumerated all of them yet. We note that the pathsplitwidth of H is exactly the *linear branch-width* [34] of the dual hypergraph of H, but we introduced pathsplitwidth because it fits our proofs better.

For a monotone DNF φ with hypergraph H, the quantity $\mathrm{psw}(H)$ is intuitively connected to the quantity of information that an OBDD will have to remember when evaluating φ following any variable ordering, which we will formalize via dncpi-sets. This being said, the definition of pathsplitwidth is also reminiscent of that of pathwidth, and we can indeed connect the two (up to a factor of the arity):

▶ **Lemma 18.** *For any hypergraph $H = (V, E)$, we have* $\mathrm{pw}(H) \leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$.

Proof sketch. From a variable ordering $\mathbf{v}$, we construct a path decomposition of H by creating $|V|$ bags in sequence, each of which containing v_i plus $\bigcup \mathrm{Split}_i(\mathbf{v}, H)$. The width is $\leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$, and we check the two conditions of path decompositions. First, each hyperedge of H is contained in a bag where it is split. Second, each vertex v_i occurs in the corresponding bag b_i and at all positions where the edges containing v are split, which forms a segment of $\mathbf{v}$: thus, the connectedness condition of tree decompositions is respected. ◀

Thanks to Lemma 18, it suffices to show that an OBDD for φ has width $\geqslant 2^{\left\lfloor \frac{\mathrm{psw}(\varphi)}{(a \times d)^2} \right\rfloor}$, which we will do in the rest of this section.

dncpi-sets. To show this lower bound, we use the technical tool of *dncpi-sets* [1,7]. We recall the definitions here, adapting the notation slightly. Remember that our monotone DNFs are assumed to be minimized. Note that dncpi-sets are reminiscent of *subfunction width* in [15] (see Theorem 17 in [15]), but the latter notion is only defined for graph CNFs.

▶ **Definition 19** ([1, Definition 6.4.6])**.** Given a monotone DNF φ on variables V, a *disjoint non-covering prime implicant set* (dncpi-set) of φ is a set S of clauses of φ which:

- are pairwise disjoint: for any $D_1 \neq D_2$ in S, we have $D_1 \cap D_2 = \emptyset$.
- are *non-covering* in the following sense: for any clause D of φ, if $D \subseteq \bigcup S$, then $D \in S$.

The *size* of S is the number of clauses that it contains.

Given a variable ordering $\mathbf{v}$ of V, we say that $\mathbf{v}$ *shatters* a dncpi-set S if there exists $1 \leqslant i \leqslant |V|$ such that $S \subseteq \mathrm{Split}_i(\mathbf{v}, H)$, where H is the hypergraph of φ.

Observe the analogy between shattering and splitting, which we will substantiate below. We recall the main result on dncpi-sets:

▶ **Lemma 20** ([1, Lemma 6.4.7]). *Let φ be a monotone DNF on variables V and $n \in \mathbb{N}$. Assume that, for every variable ordering $\mathbf{v}$ of V, there is some dncpi-set S of φ with $|S| \geqslant n$, such that $\mathbf{v}$ shatters S. Then any OBDD for φ has width $\geqslant 2^n$.*

Proof sketch. Considering the point at which the dncpi-set is shattered, the OBDD must remember exactly the status of each clause of the set: any valuation that satisfies a subset of these clauses gives rise to a different continuation function. This is where we use the fact that the DNF is monotone: it ensures that we can freely choose a valuation of the variables that do not occur in the dncpi-set without making the formula true. ◀

Concluding the proof. We conclude the proof of Theorem 15 by showing that any variable ordering of the variables of a monotone DNF φ shatters a dncpi-set of the right size. The formal statement is as follows, and it is the last result to prove:

▶ **Lemma 21.** *Let φ be a monotone DNF, H its hypergraph, and $\mathbf{v}$ an enumeration of its variables. Then there is a dncpi-set S of φ shattered by $\mathbf{v}$ such that $|S| \geqslant \left\lfloor \frac{\mathrm{psw}(H)}{(\mathrm{arity}(H)\times\mathrm{degree}(H))^2} \right\rfloor$.*

We prove this result in the rest of the section. Our goal is to construct a dncpi-set, which intuitively consists of clauses that are disjoint and which do not cover another clause. We can do so by picking clauses sufficiently "far apart". Let the *exclusion graph* of $H = (V, E)$ be the graph on E where two edges $e \neq e'$ are adjacent if there is an edge e'' of E with which they both share a node: this is in particular the case when e and e' intersect as we can take $e'' := e$. Formally, the exclusion graph is $G_H = (E, \{\{e, e'\} \in E^2 \mid e \neq e' \wedge \exists e'' \in E, (e \cap e'') \neq \emptyset \wedge (e' \cap e'') \neq \emptyset\})$. In other words, two hyperedges are adjacent in G_H iff they are different and are at distance at most 4 in the incidence graph of H.

Remember that an *independent set* in the graph G_H is a subset S of E such that no two elements of S are adjacent in G_H. The definition of G_H then ensures:

▶ **Lemma 22.** *For any monotone DNF φ, letting H be its hypergraph, any independent set of the exclusion graph G_H is a dncpi-set of φ.*

In other words, our goal is to compute a large independent set of the exclusion graph. To do this, we will use the following straightforward lemma about independent sets:

▶ **Lemma 23.** *Let $G = (V, E)$ be a graph and let $V' \subseteq V$. Then G has an independent set $S \subseteq V'$ of size at least $\left\lfloor \frac{|V'|}{\mathrm{degree}(G)+1} \right\rfloor$.*

Moreover, we can bound the degree of G_H using the degree and arity of H:

▶ **Lemma 24.** *Let H be a hypergraph. Then* $\mathrm{degree}(G_H) \leqslant (\mathrm{arity}(H) \times \mathrm{degree}(H))^2 - 1$.

Proof sketch. The bound on the arity and degree of H implies a bound on the number of edges that can be at distance $\leqslant 4$ of another edge in the incidence graph of H, hence bounding the degree of the exclusion graph. ◀

We are now ready to conclude the proof of Lemma 21:

Proof of Lemma 21. Let φ be a monotone DNF, $H = (V, E)$ its hypergraph, and $\mathbf{v}$ an enumeration of its variables. By definition of pathsplitwidth, there is $v_i \in V$ such that, for $E' := \mathrm{Split}_i(\mathbf{v}, H)$, we have $|E'| \geqslant \mathrm{psw}(H)$. Now, by Lemma 23, G_H has an independent set $S \subseteq E'$ of size at least $\left\lfloor \frac{|E'|}{\mathrm{degree}(G_H)+1} \right\rfloor$ which is $\geqslant \left\lfloor \frac{\mathrm{psw}(H)}{(\mathrm{arity}(H)\times\mathrm{degree}(H))^2} \right\rfloor$ by Lemma 24. Hence, S is a dncpi-set by Lemma 22, has the desired size, and is shattered since $S \subseteq E'$. ◀

Combining this result with Lemma 18 and Lemma 20 concludes the proof of Theorem 15.

From DNFs to CNFs. We now argue that Theorem 15 also holds for monotone CNFs. Let φ be a monotone CNF, $a := \text{arity}(\varphi)$ and $d := \text{degree}(\varphi)$, and suppose for a contradiction that there is an OBDD O for φ of width $< 2^{\left\lfloor \frac{\text{pw}(\varphi)}{a^3 \times d^2} \right\rfloor}$. Consider the monotone DNF φ' built from φ by replacing each $\wedge$ by a $\vee$ and each $\vee$ by a $\wedge$. Now, let O' be the OBDD built from O by replacing the label $b \in \{0, 1\}$ of each edge by $1 - b$, and replacing the label b of each leaf by $1 - b$. It is clear, by De Morgan's laws, that O' is an OBDD for φ' of size $< 2^{\left\lfloor \frac{\text{pw}(\varphi)}{a^3 \times d^2} \right\rfloor}$, which contradicts Theorem 15 applied to monotone DNFs.

6 Lower Bounds on d-SDNNFs

In the previous section, we have shown that *pathwidth* measures how concisely an OBDD can represent a monotone DNF or CNF formula with bounded degree and arity. In this section, we move from OBDDs to (d-)SDNNFs, and show that *treewidth* plays a similar role to pathwidth in this setting. Formally, we show the following analogue of Theorem 15:

▶ **Theorem 25.** *Let φ be a monotone DNF (resp., monotone CNF), let $a := \text{arity}(\varphi)$ and $d := \text{degree}(\varphi)$. Then any d-SDNNF (resp., SDNNF) for φ has size $\geqslant 2^{\left\lfloor \frac{\text{tw}(\varphi)}{3 \times a^3 \times d^2} \right\rfloor} - 1$.*

Combined with Theorem 5 (or with existing results specific to CNF formulas such as [14, Corollary 1]), this yields an analogue of Corollary 16. However, its statement is less neat: unlike OBDDs, (d-)SDNNFs have no obvious notion of width, so the lower bound above refers to size rather than width, and it does not exactly match our upper bound. We obtain:

▶ **Corollary 26.** *For any constant c, for any monotone DNF (resp., monotone CNF) φ with arity and degree bounded by c, there is a d-SDNNF for φ having size $|\varphi| \times 2^{O(\text{tw}(\varphi))}$, and any d-SDNNF (resp., SDNNF) for φ has size $2^{\Omega(\text{tw}(\varphi))}$.*

Our proof of Theorem 25 will follow the same overall structure as in the previous section. We present the proof for monotone DNFs and d-DNNFs: see the extended version [8] for the extension to monotone CNFs and SDNNFs. Recall that d-SDNNFs are structured by *v-trees*, which generalize variable orders. We first introduce *treesplitwidth*, a width notion that measures the performance of a v-tree by counting how many clauses it splits; and we connect treesplitwidth to treewidth. We use again dncpi-sets, and argue that a d-SDNNF structured by a v-tree must shatter a dncpi-set whose size follows the treesplitwidth of the v-tree. We then show that shattering a dncpi-set forces d-SDNNFs to be large: instead of the easy OBDD result of the previous section (Lemma 20), we will need a much deeper result of Pipatsrisawat and Darwiche [36, Theorem 3], rephrased in the setting of communication complexity by Bova, Capelli, Mengel, and Slivovsky [13].

Note that [13], by a similar approach, shows an exponential lower bound on the size of d-SDNNF which is reminiscent of ours. However, their bound again applies to one well-chosen family of Boolean functions; our contribution is to show a general lower bound. In essence, our result is shown by observing that the family of functions used in their lower bound occurs "within" any bounded-degree, bounded-arity monotone DNF. Also note that a result similar to the lower bound of Corollary 26 is proven by Capelli [19, Corollary 6.35] as an auxiliary statement to separate structured DNNFs and FBDDs. The result uses *MIM-width*, but Theorem 4.2.5 of [40], as degree and arity are bounded, implies that we could rephrase it to treewidth; further, the result assumes arity-2 formulas, but it could be extended to arbitrary arity as in [20, Theorem 12]. More importantly, the result applies only to monotone CNFs and not to DNFs .

Treesplitwidth. Informally, treesplitwidth is to v-trees what pathsplitwidth is to variable orders: it bounds the "best performance" of any v-tree.

▶ **Definition 27.** Let $H = (V, E)$ be a hypergraph, and T be a v-tree over V. For any node n of T, we define $\mathrm{Split}_n(T, H)$ as the set of hyperedges e of H that contain both a variable in T_n and one outside T_n (recall that T_n denotes the subtree of T rooted at n). Formally: $\mathrm{Split}_n(T, H) := \{e \in E \mid \exists v_\mathrm{i} \in \mathsf{Leaves}(T_n)$ and $\exists v_\mathrm{o} \in \mathsf{Leaves}(T \setminus T_n)$ such that $\{v_\mathrm{i}, v_\mathrm{o}\} \subseteq e\}$.

The *treesplitwidth* of T relative to H is $\mathrm{tsw}(T, H) := \max_{n \in T} |\mathrm{Split}_n(T, H)|$. The *treesplitwidth* $\mathrm{tsw}(H)$ of H is then the minimum of $\mathrm{tsw}(T, H)$ over all v-trees T of V.

Again, the treesplitwidth of H is exactly the *branch-width* [38] of the dual hypergraph of H, but treesplitwidth is more convenient for our proofs. As with pathsplitwidth and pathwidth (Lemma 18), we can bound the treewidth of a hypergraph by its treesplitwidth:

▶ **Lemma 28.** *For any hypergraph $H = (V, E)$, we have $\mathrm{tw}(H) \leqslant 3 \times \mathrm{arity}(H) \times \mathrm{tsw}(H)$.*

Proof sketch. We construct a tree decomposition from a v-tree T: it has same skeleton as T, its leaf bags contain the corresponding variable in the v-tree, and its internal bags contain the split at this v-tree node unioned with the split at the child nodes. This is indeed a tree decomposition because each non-singleton edge is split, and the nodes of the v-tree where a vertex of H occurs always form a connected subtree. ◀

Moreover, using the same techniques that we used in the last section, we can show the analogue of Lemma 21. Specifically, given a monotone DNF φ on variables V, a v-tree T over V, and a dncpi-set S of φ, we say that T *shatters* S if there is a node n in T such that $S \subseteq \mathrm{Split}_n(T, \varphi)$. We now show that any v-tree over V must shatter a large dncpi-set (depending on the treewidth, degree, and arity):

▶ **Lemma 29.** *Let φ be a monotone DNF, H its hypergraph, and T be a v-tree over its variables. Then there is a dncpi-set S of φ shattered by T such that $|S| \geqslant \left\lfloor \frac{\mathrm{tsw}(H)}{(\mathrm{arity}(H) \times \mathrm{degree}(H))^2} \right\rfloor$.*

Proof sketch. The proof is just like that of Lemma 21, except with the new definition of split on v-trees. In particular, we use Lemmas 22, 23, and 24. ◀

Hence, to prove Theorem 25, the only missing ingredient is a lower bound on the size of d-SDNNFs that shatter large dncpi-sets. Specifically, we need an analogue of Lemma 20:

▶ **Lemma 30.** *Let φ be a monotone DNF on variables V and $n \in \mathbb{N}$. Assume that, for every v-tree T over V, there is some dncpi-set S of φ with $|S| \geqslant n$, such that T shatters S. Then any d-SDNNF for φ has size $\geqslant 2^n - 1$.*

We will prove Lemma 30 in the rest of this section using a recent lower bound by Bova, Capelli, Mengel, and Slivovsky [13]. They bound the size of any d-SDNNF for the *set intersection* function, defined as $\mathrm{SINT}_n := (x_1 \wedge y_1) \vee \ldots \vee (x_n \wedge y_n)$. This bound is useful for us: a dncpi-set intuitively isolates some variables on which φ computes exactly SINT_n:

▶ **Lemma 31.** *Let φ be a DNF with variables V, and let $S = \{D_1, \ldots, D_n\}$ be a dncpi-set of φ where every clause has size $\geqslant 2$. Pick two variables $x_i \neq y_i$ in D_i for each $1 \leqslant i \leqslant n$, and let $V' := \{x_1, y_1, \ldots, x_n, y_n\}$. Then there is a partial valuation ν of V with domain $V \setminus V'$ such that $\nu(\varphi) = \mathrm{SINT}_n$.*

Proof sketch. The valuation ν sets to 1 the variables V'' which are in $\bigcup S$ but not in V', and sets to 0 all remaining variables. This amounts to discarding the clauses not in the dncpi-set, and discarding the variables of V'' in the dncpi-set: what remains of the DNF is then precisely SINT_n. Note that this result relies on monotonicity, and on the fact that φ is a DNF. (However, in the full version [8], we show a dual result for monotone CNF.) ◀

This observation allows us to leverage the bound of [13] on the size of d-SDNNFs that compute SINT_n, assuming that they are structured by an "inconvenient" v-tree:

▶ **Proposition 32** ([13, Proposition 14]). *Let $X_n = \{x_1, \ldots, x_n\}$ and $Y_n = \{y_1, \ldots, y_n\}$ for $n \in \mathbb{N}$, and let T be a v-tree over $X_n \sqcup Y_n$ such that there exists a node $n \in T$ with $X_n \subseteq \mathsf{Leaves}(T_n)$ and $Y_n \subseteq \mathsf{Leaves}(T \setminus T_n)$. Then any d-SDNNF structured by T computing SINT_n has size $\geqslant 2^n - 1$.*

In our setting, an "inconvenient" v-tree for a dncpi-set is one that shatters it: each clause of the dncpi-set is then partitioned in two non-empty subsets where we can pick x_i and y_i for Lemma 31. Hence, when every v-tree shatters a large dncpi-set of φ, Proposition 32 allows us to deduce the lower bound on the size of every d-SDNNF for φ. We have thus shown Lemma 30, and this concludes the proof of Theorem 25 (in the DNF case).

7 Application to Query Lineages

In this section, we adapt the lower bound of the previous section to the computation of query lineages on relational instances. Like in [7], for technical reasons, we must assume a graph signature. We first recall some preliminaries and then state our result.

Preliminaries. We fix a *graph signature* σ of relation names and arities in $\{1, 2\}$, with at least one relation of arity 2. An *instance* I on σ is a finite set of *facts* of the form $R(a_1, \ldots, a_n)$ for n the arity of R; we call $a_1, \ldots, a_n$ *elements* of I. An instance I' is a *subinstance* of I if the facts of I' are a subset of those of I. The *Gaifman graph* of I has the elements of I as vertices and has one edge between each pair of elements that co-occur in some fact of I. The *treewidth* $\mathrm{tw}(I)$ of I is that of its Gaifman graph.

A *Boolean conjunctive query* (CQ) is an existentially quantified conjunction of *atoms* of the form $R(x_1, \ldots, x_n)$ where the x_i are *variables*. A *UCQ* is a disjunction of CQs, and a $\mathrm{UCQ}^{\neq}$ also allows atoms of the form $x \neq y$. A $\mathrm{UCQ}^{\neq}$ is *connected* if the Gaifman graph of each disjunct (seen as an instance, and ignoring $\neq$-atoms) is connected. For instance, letting σ_R consist of one arity-2 relation R, the following connected $\mathrm{UCQ}^{\neq}$ tests if there are two facts that share one element: $Q_{\mathrm{p}} : \exists xyz\ (R(x, y) \vee R(y, x))) \wedge (R(y, z) \vee R(z, y)) \wedge x \neq z$. (While Q_{p} is not given as a disjunction of CQs, it can be rewritten to one using distributivity.)

The *lineage* of a $\mathrm{UCQ}^{\neq}$ Q over I is a Boolean formula $\varphi(Q, I)$ on the facts of I that maps each Boolean valuation $\nu : I \to \{0, 1\}$ to 1 or 0 depending on whether I_ν satisfies Q or not, where $I_\nu := \{F \in I \mid \nu(F) = 1\}$. The lineage intuitively represents which facts of I suffice to satisfy Q. Lineages are useful to evaluate queries on *probabilistic databases* [39]: we can obtain the probability of the query from an OBDD or d-DNNF representing its lineage.

Problem statement. We study when query lineages can be computed efficiently in *data complexity*, i.e., as a function of the input instance, with the query being fixed. A first question asks which *queries* have *tractable lineages* on all instances: Jha and Suciu [32, Theorem 3.9] showed that *inversion-free* $\mathrm{UCQ}^{\neq}$ queries admit OBDD representations in this sense, and Bova and Szeider [16, Theorem 5] have recently shown that $\mathrm{UCQ}^{\neq}$ queries with inversions do not even have tractable d-SDNNF lineages. A second question asks which *instance classes* ensure that *all queries* have tractable lineages on them. This was studied for OBDD representations in [7]: bounded-treewidth instances have tractable OBDD lineage representations for any MSO query ([7, Theorem 6.5], using [32]); conversely there are *intricate* queries (a class of connected $\mathrm{UCQ}^{\neq}$ queries) whose lineages never have tractable OBDD representations in the instance treewidth [7, Theorem 8.7]. The query Q_{p} above is an example of an intricate query on the signature σ_R (refer to [7, Definition 8.5] for the formal definition of intricate queries). This result shows that we must bound instance treewidth for all queries to have tractable OBDDs, but leaves the question open for more expressive lineage representations.

Result. Our bound in the previous section allows us to extend Theorem 8.7 of [7] from OBDDs to d-SDNNFs, yielding the following:

▶ **Theorem 33.** *There is a constant $d \in \mathbb{N}$ such that the following is true. Let σ be an arity-2 signature, and Q a connected* $\mathrm{UCQ}^{\neq}$ *which is intricate on σ. For any instance I on σ, any d-SDNNF representing the lineage of Q on I has size $2^{\Omega(\mathrm{tw}(I)^{1/d})}$.*

Proof sketch. As in [7], we use a result of Chekuri and Chuzhoy [21] to show that the Gaifman graph of I has a degree-3 topological minor S of treewidth $\Omega(\mathrm{tw}(I)^{1/d})$ for some constant $d \in \mathbb{N}$; we also ensure that S has sufficiently high *girth* relative to Q. We focus on a subinstance I' of I that corresponds to S: this suffices to show our lower bound, because we can always compute a tractable representation of $\varphi(Q, I')$ from one of $\varphi(Q, I)$. Now, we can represent $\varphi(Q, I')$ as a minimized DNF ψ by enumerating its minimal matches: ψ has constant arity because the number of atoms of Q is fixed, and it has constant degree because S has constant degree and Q is connected. Further, as Q is intricate and I' has high girth relative to Q, we can ensure that this DNF has treewidth $\Omega(\mathrm{tw}(I'))$. We conclude by Theorem 25: d-SDNNFs representing $\varphi(Q, I')$, hence $\varphi(Q, I)$, have size $2^{\Omega(\mathrm{tw}(I)^{1/d})}$. ◀

To summarize, given an instance family $\mathcal{I}$ satisfying the constructibility requirement of Theorem 8.1 of [7], there are two regimes: (i.) $\mathcal{I}$ has bounded treewidth and then all MSO queries have d-SDNNF lineages on instances of $\mathcal{I}$ that are computable in linear time; or (ii.) the treewidth is unbounded and then there are $\mathrm{UCQ}^{\neq}$ queries (the intricate ones) whose lineages on instances of $\mathcal{I}$ have no d-SDNNF representations polynomial in the instance size.

8 Conclusion

We have shown tight connections between structured circuit classes and width measures on circuits. We constructively rewrite bounded-treewidth circuits to d-SDNNFs in time linear in the circuit and singly exponential in the treewidth, and show matching lower bounds for arbitrary monotone CNFs or DNFs under degree and arity assumptions; we also show a lower bound for pathwidth and OBDDs. Our results have applications to rich query evaluation: probabilistic query evaluation, computation of lineages, enumeration, etc.

Our work also raises a number of open questions. First, the d-SDNNF obtained in the proof of Theorem 5 does *not* respect the definition of a *sentential decision diagram* (SDD) [24]. Can this be fixed, and Theorem 5 extended to SDDs? Or is it impossible, which could solve the open question [11] of separating SDDs and d-SDNNFs? Second, can we weaken the hypotheses of bounded degree and arity in Corollaries 16 and 26, and can we rephrase the latter to a notion of (d-)SDNNF width to match more closely the statement of the former? Last, Section 7 shows that d-SDNNF representations of the lineages of intricate queries are exponential in the treewidth; we conjecture a similar result for pathwidth and OBDDs, but this would require a pathwidth analogue of the minor extraction results of [21].

Acknowledgments. We acknowledge Chandra Chekuri for his helpful comments at `https://cstheory.stackexchange.com/a/38943/`, as well as Florent Capelli for pointing out the connection to [19, Corollary 6.35] and [40].

References

1 Antoine Amarilli. *Leveraging the Structure of Uncertain Data*. PhD thesis, Télécom ParisTech, 2016.

2 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *ICALP*, 2017.

3 Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits. In *ICDT*, 2017.

4 Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits (Extended version). *CoRR*, abs/1612.04203, 2017. Extended version of [3].

5 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances (extended version). *CoRR*, abs/1511.08723, 2015. Extended version of [6].

6 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, July 2015.

7 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: limits and extensions. In *PODS*, 2016.

8 Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Connecting width and structure in knowledge compilation (extended version). *CoRR*, abs/1709.06188, 2017.

9 Paul Beame and Vincent Liew. New limits for knowledge compilation and applications to exact model counting. In *UAI*, 2015.

10 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6), 1996.

11 Simone Bova. SDDs are exponentially more succinct than OBDDs. In *AAAI*, 2016.

12 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. A strongly exponential separation of DNNFs from CNF formulas. *CoRR*, abs/1411.1995, 2015.

13 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge compilation meets communication complexity. In *IJCAI*, 2016.

14 Simone Bova and Friedrich Slivovsky. On compiling structured CNFs to OBDDs. In *CSR*, 2015.

15 Simone Bova and Friedrich Slivovsky. On compiling structured CNFs to OBDDs. *TCS*, 61(2), 2017.

16 Simone Bova and Stefan Szeider. Circuit treewidth, sentential decision, and query compilation. In *PODS*, 2017.

17 Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3), 1992.

18 Andrea Calì, Florent Capelli, and Igor Razgon. Non-FPT lower bounds for structural restrictions of decision DNNF. *CoRR*, abs/1708.07767, 2017.

19 Florent Capelli. *Structural restrictions of CNF-formulas: applications to model counting and knowledge compilation*. PhD thesis, Université Paris-Diderot, 2016.

20 Florent Capelli. Understanding the complexity of #SAT using knowledge compilation. In *LICS*, 2017.

21 Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. In *STOC*, 2014.

22 Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Applied Non-Classical Logics*, 11(1-2), 2001.

23 Adnan Darwiche. A differential approach to inference in Bayesian networks. *JACM*, 50(3), 2003.

24 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, 2011.

25 Srinivas Devadas. Comparing two-level and ordered binary decision diagram representations of logic functions. *IEEE TCAD*, 12(5), 1993.

26 Andrea Ferrara, Guoqiang Pan, and Moshe Y Vardi. Treewidth in verification: Local vs. global. In *LPAR*, 2005.

27 Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *TPLP*, 15(3), 2015.

28 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.

29 Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *J. Combinatorial Theory, Series B*, 99(1), 2009.

30 Abhay Kumar Jha, Dan Olteanu, and Dan Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, 2010.

31 Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. In *ICDT*, 2011.

32 Abhay Kumar Jha and Dan Suciu. On the tractability of query compilation and bounded treewidth. In *ICDT*, 2012.

33 Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society. Series B*, 1988.

34 Joakim Alme Nordstrand. Exploring graph parameters similar to tree-width and path-width. Master's thesis, University of Bergen, 2017.

35 Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, 2008.

36 Knot Pipatsrisawat and Adnan Darwiche. A lower bound on the size of decomposable negation normal form. In *AAAI*, 2010.

37 Igor Razgon. On OBDDs for CNFs of bounded treewidth. In *KR*, 2014.

38 Neil Robertson and P.D Seymour. Graph minors. x. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153 – 190, 1991.

39 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

40 Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, 2012.