On the Tractability of Query Compilation and Bounded Treewidth

Abhay Jha
Computer Science and Engineering
University of Washington
Seattle, WA 98195–2350
abhaykj@cs.washington.edu

Dan Suciu
Computer Science and Engineering
University of Washington
Seattle, WA 98195–2350
suciu@cs.washington.edu

ABSTRACT

We consider the problem of computing the probability of a Boolean function, which generalizes the model counting problem. Given an OBDD for such a function, its probability can be computed in linear time in the size of the OBDD. In this paper we investigate the connection between treewidth and the size of the OBDD. Bounded treewidth has proven to be applicable to many graph problems, which are NP-hard in general but become tractable on graphs with bounded treewidth. However, it is less well understood how bounded treewidth can be used for the probability computation problem of a Boolean function. We introduce a new notion of treewidth of a Boolean function, called the *expression treewidth*, as the smallest treewidth of any DAG-expression representing the function. Our new notion of bounded treewidth includes some previously known tractable cases: all read-once Boolean functions, and all functions having a bounded treewidth of the primal graph or of the incidence graph also have a bounded expression treewidth. We show that bounded expression treewidth implies the existence of a polynomial size OBDD, and that bounded expression pathwidth implies the existence of a constant-width OBDD. We also show a converse of the latter result: constant-width OBDD imply bounded expression pathwidth. We then study the implications of these results to query compilation, where the Boolean function is the lineage of a fixed query on varying input databases. We give a syntactic characterizations of all $UCQ^{\neq}$ queries that admit a polynomial size OBDD, showing that these are precisely *inversion-free* queries with unrestricted use of $\neq$. It was previously known that inversion-free queries characterize precisely those UCQ queries that have a polynomial size OBDD, and that these also have a constant width OBDD: in contrast, inversion-free queries with $\neq$ have polynomial-width OBDD, thus using the full power of OBDD. Finally, we show that in the case of UCQ, the four classes studied in this paper collapse: bounded expression pathwidth, bounded expression treewidth, constant-width OBDD, and polynomial size OBDD.

ICDT 2012, March 26–30, 2012, Berlin, Germany.

Categories and Subject Descriptors

H.2.3 [**DATABASE MANAGEMENT**]: Languages—*Query Languages*; F.1.1 [**COMPUTATION BY ABSTRACT DEVICES**]: Models of Computation; G.3 [**Probability and statistics**]: Probabilistic Algorithms

General Terms

Algorithms, Management, Theory

Keywords

Probabilistic databases, Knowledge compilation, Binary Decision Diagrams, OBDD, Treewidth

1. INTRODUCTION

In this paper we study the connection between the treewidth of a Boolean function, and the size of an OBDD for that function. Our main motivation comes from query evaluation on probabilistic database, which, at its core, consists of computing the probability of a Boolean function (namely, the query's lineage).

The *tree-width* of a graph is a measure of how tree-like a graph is. A tree has a tree-width equal to one, while a complete graph, or complete bipartite graph has a large tree-width. Many graph problems that are hard on arbitrary graphs, become tractable over graphs with a bounded tree-width [13, 4]. This also holds for probabilistic inference problem in graphical models. In fact, *all* exact probabilistic inference algorithms on graphical models known in the literature run in time that is exponential in the tree-width [22, 7, 9], so, for all practical purposes, bounded tree-width characterizes tractability in graphical models.

The probabilistic inference for Boolean functions, which concerns us in this paper, is a special case of inference in graphical models, and has quite distinct characteristics. The problem asks for the probability that a Boolean expression is true, if each of its variables is set to true independently, with a given probability. It generalizes model counting, and is #P-hard, even for positive 2CNF expressions [27]. The extent to which bounded tree-width can be used in this context is less well understood. In the *primal graph* of a CNF expression nodes represent variables, and edges represent pairs of variables that co-occur in a clause; it follows (from algorithms on graphical models) that, if the tree-width of the

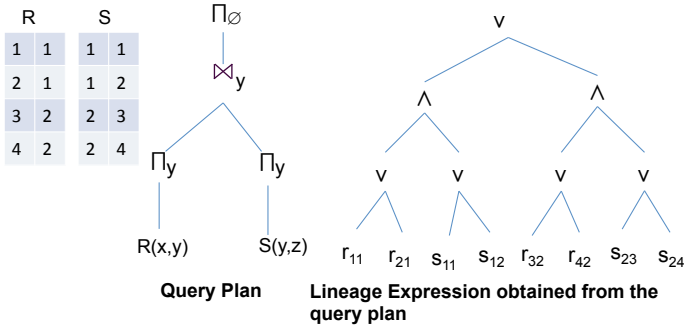

Figure 1: The lineage of the query $q_{rs} = R(x, y), S(y, z)$ on a small database instance; s_{ij} denotes the tuple $S(i, j)$ and similarly for the others. On *any* database instance, the lineage of q_{rs} is a read-once expression [20], hence, the expression tree-width is 1. Its OBDD width is also 1, hence we show that its expression path-width is ≤ 5; on the other hand, the tree width of the incidence graph for either CNF or DNF is unbounded. We also illustrate the query plan that we used to compute the lineage.

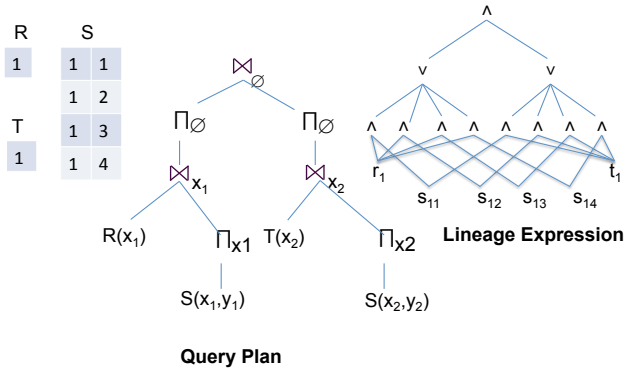

Figure 1: The lineage of the query $q_{rsst} = R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2)$ on a small database. Notice that each variable occurs only once, and as a consequence the expression is a DAG. On *any* database instance, the lineage of q_{rsst} has an OBDD of width $\leq 2^4 = 16$ [18], hence the expression path-width is ≤ 80. We also show the query plan that we used to compute the lineage.

primal graph is bounded, then the probability of the Boolean function can be computed in PTIME. But this notion of tree-width leaves out some very simple tractable cases, like a single clause, $X_1 \vee \ldots \vee X_n$, whose probability can be computed in linear time[1] yet its primal graph is a complete graph and has tree-width n. Fischer et al. proved a stronger connection [11]. They defined the *incidence graph* of a CNF expression to be the bipartite graph with one node for each variable and one node for each clause, and edges connect the variables with the clauses where they appear. They proved that, if the tree-width of the incidence graph is bounded, then the probability of the Boolean function can be computed in PTIME[2]. The dual result also holds: if the tree-width of the incidence graph defined for the DNF is bounded, then the probability can also be computed in PTIME. However, this notion of tree-width is also insufficient, because it leaves out a large class of Boolean expressions. A *read-once Boolean* expressions [15, 12] is an expression using connectives $\wedge, \vee, \neg$ where each Boolean variable may occur only once. The probability of a read-once Boolean expression can be computed in linear time in its size, because here the laws of independence hold, e.g. $\Pr(E_1 \wedge E_2) = \Pr(E_1) \cdot \Pr(E_2)$; an example of a read-once Boolean expression is in Fig. 1. However, the incidence graphs of both the CNF and the DNF representation of a read-once expression can have arbitrarily large tree-width. Since read-once expressions are of particular interest in probabilistic databases [20, 25, 23], the tree-width of the incidence graph is too weak for identifying tractable cases in these applications.

A concept that captures many tractable instances of model counting and probability computation for Boolean functions are Ordered Binary Decision Diagrams (OBDDs) [28]. An OBDD is a special case of a branching program: it is a rooted DAG where each internal node is labeled with a Boolean variable and has two outgoing edges, labeled 0 and 1, and has two leaves, labeled 0 and 1 respectively; furthermore, it is required that every path from the root to a leaf visits the Boolean variables in the same order. Given an OBDD for a Boolean function, one can compute its probability in linear time in the size of the OBDD. Thus, Boolean functions that have an OBDD of polynomial size are tractable. This class includes all read-once expressions: each read-once expression has an OBDD of width 1 (meaning that each variable occurs at most once, hence the size is linear). A connection between tree-width and OBDD was established by Huang&Darwiche [16] and Ferrara et. al. [10], who proved that if the primal graph of a CNF has a bounded tree-width, then the Boolean function defined by the CNF has an OBDD of polynomial size.

Our contributions In this paper we introduce a more powerful notion of tree-width for Boolean functions, which captures a more robust class of tractable functions. An *expression DAG* is a rooted DAG whose internal nodes are labeled with $\wedge, \vee, \neg$ and whose leaves are labeled with the Boolean variables, such that each variable occurs at most once. The *expression treewidth* of a Boolean function is the smallest treewidth of any expression DAG that represents it. This notion generalizes previous notions of tree-width: both the CNF and the DNF representations of a Boolean function correspond to some expression DAG, and if the incidence graph has bounded tree-width, then the expression tree-width is bounded too. It also naturally includes read-once expressions: every read-once Boolean expression has an expression treewidth 1, because it is given by an expression DAG that is a tree. Similarly, we define the *expression pathwidth* of a Boolean function as the smallest pathwidth of any expression representing it. For example, consider the two Boolean expressions shown in Fig. 1 and Fig. 1, which represent the lineages of two conjunctive queries, q_{rs} and q_{rsst} respectively. It is known [20] that the lineage of q_{rs} is always a read-once expression, for any input database; hence its expression tree-width is 1. It is far less obvious (but we

[1] $1 - (1 - \Pr(X_1)) \cdots (1 - \Pr(X_n))$.

[2] They only proved that model counting can be solved in PTIME, but their algorithm generalizes immediately to probabilistic inference.

will show below) that the lineage of q_{rs} on *any* database instance has an expression path-width ≤ 5, and, similarly, the lineage of q_{rsst} has an expression path-width ≤ 80.

In this paper we prove several results connecting expression tree- (or path-) width to OBDD, in three different settings: for unrestricted Boolean functions, for Boolean functions that are lineages of Unions of Conjunctive Queries with $\neq$ ($UCQ^{\neq}$), and for Unions of Conjunctive Queries (UCQ).

Results relating expression-width to OBDD Our first result is the following: if the expression pathwidth of a Boolean function is $< k$, then there exists an OBDD for it whose width is $\leq 2^{(k+1)2^{k+1}}$; in particular, the size of the OBDD is linear in the number of Boolean variables. We also show that, if the expression tree-width is bounded, then there exists an OBDD whose size is polynomial in the number of variables. Note that the former result does not imply the latter, unlike in prior work by Ferrara et al. [10]. They prove that if the primal graph has pathwidth $\leq k$, then there exists an OBDD of size $O(n2^k)$: in any graph of tree-width t, the path width is bounded by $k = O(t \log n)$, hence, if the tree-width is bounded, then the OBDD has a polynomial size because $O(n2^{t \log n}) = n^{O(1)}$. In our setting, however, the path-width occurs in a double exponent, preventing us from applying the same argument. Instead, we prove both results on expression path-width and tree-width together, using a common technical lemma.

We also show the following converse: if a Boolean function has an OBDD of width $\leq w$, then its expression pathwidth is $\leq 5w$. In other words, the notions of bounded expression pathwidth and bounded OBDD width coincide. It is known that every read-once Boolean expression has an OBDD of width 1: our result implies that any read-once expression has an expression path-width ≤ 5. For another illustration, it is known from [18] that the lineage of q_{rsst} in Fig. 1 on *any* database has an OBDD of width $\leq 2^4 = 16$: therefore its expression path-width is ≤ 80. Our results are summarized in the first diagram of Fig. 2.

Application to Query Compilation While our main motivation was to apply these results to query compilation, it turned out that query compilation actually helped us better understand the relationships between expression path/tree width and OBDDs. We are interested in the following problem: for a fixed query, determine whether, for any input database, the Boolean function representing the lineage of this query on some arbitrary database has a bounded expression path/tree width, or a polynomial size OBDD. In prior work [18] we have shown that a Union of Conjunctive Queries, *UCQ*, has a polynomial size OBDD iff it is *inversion-free*, here denoted *IF*; we also proved that every query *IF* has constant width OBDD. It follows that, when restricted to lineages of *UCQ* queries, these three classes collapse: bounded expression path/tree width, and polynomial size OBDD.

Next, we looked at Unions of Conjunctive Queries extended with $\neq$, $UCQ^{\neq}$, and found that their lineage expressions have more interesting properties. First we proved that a query in $UCQ^{\neq}$ has a polynomial-size OBDD iff, after removing $\neq$, the query is inversion free. Thus, $IF^{\neq}$ characterizes the class of queries with polynomial size OBDD. However, unlike *IF*, which have constant-width OBDD, these queries have polynomial-width OBDD, and therefore use the full power of OBDD. In fact, we prove something stronger: we show that a particular query in $IF^{\neq}$, q_{nr}, has no bounded expression treewidth. This is a result of interest beyond query compilation, because it separates Boolean functions of bounded expression treewidth from those with a polynomial size OBDD: by giving this separation through a query, we obtain a very simple formulation of the result. Moving lower in the hierarchy, we seek to separate bounded expression pathwidth and bounded expression treewidth. We give a Boolean function, bt_m, that separates these two classes, but we could not find a query in $UCQ^{\neq}$ whose lineage separates them. We conjecture that, when restricted to query lineages, these classes collapse. We further describe a syntactic class of queries, $H^{\neq}$, whose lineage have bounded expression pathwidth.

Discussion To the best of our knowledge, ours are the first results connecting the tree(path)-width of the expression DAG of a Boolean function to the size of the OBDD. Related is a very general result by Courcelle et. al. [6] that implies that probabilistic inference is tractable given a tree decomposition of an expression DAG, and a more efficient algorithm for the same problem [17], with time complexity $16^t|G|$, where G is the expression DAG and t is its tree-width.

Our expression tree-width inherits a general weakness of tree-width based approaches and of OBDD: it is NP-hard to compute the tree-width of a general graph, and similarly it is NP-hard to compute an optimal OBDD. To this, expression tree-width adds another layer of difficulty, since one also needs to find the expression DAG that minimizes the tree width. In fact, we do not know if computing the expression tree-width of an arbitrary Boolean function is even decidable. However, when the Boolean function is restricted to the lineage of a $UCQ^{\neq}$ query, then in all tractable cases we also give polynomial time algorithms for computing the polynomial size OBDD.

The class $UCQ^{\neq}$ has not been studied before in the context of probabilistic databases. Olteanu and Huang study the join-free fragment of $CQ^{<}$, and characterize all queries that have a polynomial size OBDD[21]. The characterization of queries in $CQ^{<}$ or in $UCQ^{<}$ with polynomial size OBDD is open.

This paper is organized as follows. In Sect. 2 we give our results connecting expression treewidth/pathwidth with OBDD; in Sect. 3 we give the results on $UCQ^{\neq}$. The proofs for Sect. 2 are in Sect. 4, while the proofs for Sect. 3 are in Sect. 5. All other missing proofs can be found in the Appendix. We conclude in Sect. 6.

2. RESULTS ON TREEWIDTH AND OBDD

In this section, we will define formally the *expression treewidth* and give the results relating it to OBDD size/width.

2.1 Treewidth

A graph G is a pair $(V(G), E(G))$, where $E(G) \subseteq V(G) \times V(G)$. We call $V(G)$ the vertices and $E(G)$ the edges in the graph G. A *tree-decomposition* of G is a tree $T = (V(T), G(T))$, where $V(T) = \bar{Y} = \{Y_1, Y_2, \ldots\}$ is a family of subsets of $V(G)$ s.t.

1. $\bigcup_i Y_i = V(G)$
2. for every edge $(u, v) \in E(G)$, there exists an Y_i s.t. $u \in Y_i$ and $v \in Y_i$

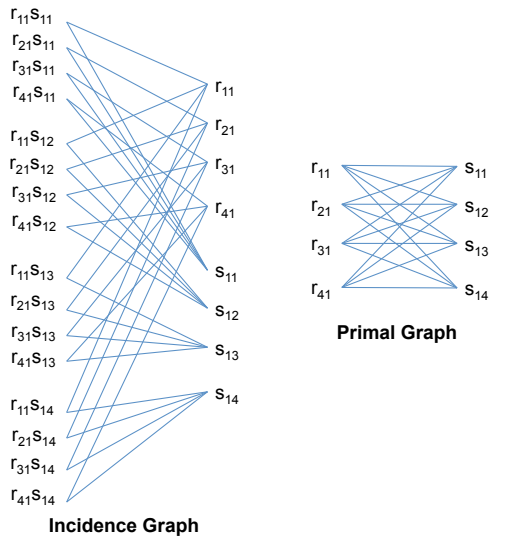

Figure 2: Primal and incidence graphs for the read-once Boolean expression $F_{mnp} = \bigvee_{j=1,n}(\bigvee_{i=1,m} r_{ij}) \wedge (\bigvee_{k=1,p} s_{jk})$, for $m = 4, n = 1, p = 4$. This expressions is precisely the lineage of $R(x,y), S(y,z)$ from Fig. 1 on the database $R = \{(i,j) \mid i \in [m], j \in [n]\}$, $S = \{(j,k) \mid j \in [n], k \in [p]\}$.

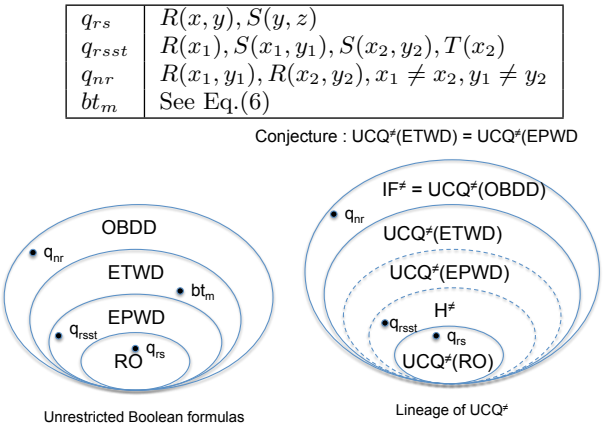

Figure 2: Relationship between tractability w.r.t RO,OBDD and the new notions of expression pathwidth and treewidth. RO=*read-once*, EPWD=*bounded expression pathwidth*, ETWD=*bounded expression tree-width*, OBDD=*polynomial-size OBDD*

3. For any $v \in V(G)$, the set $\{Y_i \mid v \in Y_i\}$ forms a connected component of T.

The *width* of the tree-decomposition is defined as $\max_i |Y_i| - 1$. The *treewidth* of G, $twd(G)$, is defined as the minimum width over all possible tree-decompositions of G. Note that all trees have treewidth 1. Analogous to treewidth, the *pathwidth*, $pwd(G)$, is the minimum width over all path-decompositions, where a path-decomposition is defined just like a tree-decomposition except that T is required to be a *path* instead of a tree. If the graph has n nodes, then $twd(G) \leq pwd(G) = O(twd(G) \cdot \log n)$. The last inequality is *nearly* tight: if G is a complete binary tree with $2^{k+1} - 1$ nodes then $pwd(G) = \lceil \frac{k}{2} \rceil$ [24]. Many problems that are intractable on general graphs become tractable over graphs of bounded treewidth [13, 4]. The problem of determining whether treewidth $< k$ was shown to be NP-complete in [1]. For fixed k though, Bodlaender [3] showed that one can construct a tree-decomposition of width k in linear time.

2.2 Expression Treewidth

Let F be a Boolean function over Boolean variables $\bar{X} = X_1, X_2, \ldots, X_n$. The problem of interest to us is :

DEFINITION 2.1. *The* probability inference problem *is : Given F and probability assignments p_i to each variable X_i, compute the probability of F,* $\Pr(F)$*, where*

$$\Pr(F) = \sum_{z:\bar{X}\to\{0,1\}, F(z)=1} \prod_{z(X_i)=0} (1-p_i) \prod_{z(X_i)=1} p_i \qquad (1)$$

Inference is $\#P - complete$, even for positive 2CNF [27].

To define the expression treewidth, we make the usual distinctions between a Boolean function, and an expression using $\wedge, \vee, \neg$ representing that function.

DEFINITION 2.2. *An* expression *E over variables $\bar{X}$ is defined by the following grammar :*

$$E ::= X_i \mid \neg E \mid E_1 \vee \ldots \vee E_m \mid E_1 \wedge \ldots \wedge E_p \qquad (2)$$

CNF and DNF are particular examples of expressions.

DEFINITION 2.3. *An expression DAG G is a rooted DAG whose internal nodes are labeled with $\wedge, \vee, \neg$, and whose leaves are labeled with Boolean variables, s.t. each variable occurs at most once. The graph* represents *the Boolean function given by the expression obtained by unfolding the DAG into a tree.*

A simple illustration of an expression DAG is in Fig. 1. Finally, the expression treewidth is:

DEFINITION 2.4. *The* expression treewidth *of a Boolean function F, $etwd(F)$, is defined as* $\min twd(G)$ *over all possible expression DAGs G representing F. Similarly, the* expression pathwidth*, $epwd(F)$, is defined as* $\min pwd(G)$.

Our definition is robust w.r.t. the choice of operators used in the grammar Eq. (2), in the following sense. Consider a different grammar:

$$E ::= X_i \mid \neg E_1 \mid E_1 \otimes E_2, \qquad \text{where } \otimes \in \mathbb{B}^{\mathbb{B}\times\mathbb{B}} \qquad (3)$$

Here $\mathbb{B}^{\mathbb{B}\times\mathbb{B}}$ is the set of all 2^4 Boolean operators over two variables. Define $etwd_*(F)$ to be $\min twd(G)$ where G ranges over all possible DAGs representing F using the extended grammar. We prove:

PROPOSITION 2.5. *For any Boolean function F we have $etwd_*(F) \leq etwd(F) \leq etwd_*(F) + 2$.*

2.3 Background: Some Tractable Functions

We explain now the connection between bounded expression treewidth and some previously known tractable functions, and start with read-once functions.

DEFINITION 2.6. *[15, 12] A Boolean expression is called* read-once *if every variable occurs at most once. A Boolean function is called* read-once *if it admits a read-once expression.*

It is known that the inference problem can be solved in linear time for a read-once expression. They are of particular interest in probabilistic databases, where an important class of queries is known to have lineage expressions that are read-once [20, 18]. Obviously, if F is a read-once function then $etwd(F) = 1$, because the read-once expression is a tree.

Next, assume that F is given as a DNF expression[3], $\bigvee_j T_j$, where each $T_j = \bigwedge_i L_i$, and each literal L_i is either a Boolean variable or its negation. We assimilate T_j with the set of Boolean variables occurring in T_j. The *primal graph* G^P, and the *incidence graph* of F are defined as

$$G^P = \big(\bar{X}, \{(X_i, X_j) \mid \exists T_k.\ X_i \in T_k \wedge X_j \in T_k\}\big)$$
$$G^I = \big(\bar{T}, \bar{X}, \{(T_k, X_i) \mid X_i \in T_k\}\big)$$

In the primal graph the nodes are variables and the edges are pairs of co-occurring variables. The incidence graph is bipartite; its nodes are the conjuncts and the variables, and its edges connected each conjuct with the variables it contains. The two graphs are of interest in our setting since bounded treewidth in either case leads to tractable inference.

PROPOSITION 2.7. *[11] The inference problem for F can be solved in time* $\min\left(2^{twd(G^P)}, 4^{twd(G^I)}\right) \cdot O(n)$

The result for primal graph follows from the classical complexity results of inference over Markov Networks [22, 7]. The result for incidence graph is due to [11]. The relationship is [19, pp. 327-328]:

PROPOSITION 2.8. *Let $m = \max_j |T_j|$, then $twd(G^I) \leq twd(G^P) + 1$ and $twd(G^P) \leq (twd(G^I) + 1) \cdot m - 1$*

Thus, a bounded treewidth of the primal graph always implies a bounded treewidth of the incidence graph; the converse holds too when the size of the conjuncts is bounded: the latter is indeed the case in query compilation.

A bounded treewidth of the incidence graph also implies a bounded expression treewidth. More precisely, for any Boolean function F, $etwd(F) \leq twd(G^I)+1$. Thus, tractability results for bounded expression treewidth are at least as strong as Prop. 2.7. However, the converse fails. In particular, read-once Boolean functions have, in general, incidence graphs with large treewidth, while their expression treewidth is $= 1$. Thus, bounded treewidth is strictly stronger.

EXAMPLE 2.9. *Consider the Boolean function $F_{mnp} = \bigvee_{j=1,n}[(\bigvee_{i=1,m} r_{ij}) \wedge (\bigvee_{k=1,p} s_{jk})]$. This Boolean function occurs naturally in query compilation, since it is the lineage of the query $R(x,y), S(y,z)$. Written in DNF it becomes $\bigvee_{i=1,m;j=1,n;k=1,p} r_{ij} \wedge s_{jk}$, thus, in the primal graph all variables $r_{1j}, \ldots, r_{mj}$ are connected to all variables $s_{j1}, \ldots, s_{jp}$: Fig. 2 shows the primal graph (on the right) and also the incidence graph (left) for the case $m = p = 4, n = 1$. More generally, if $n = 1$ and $m = p$ are arbitrary, then the treewidth of primal graph is m, and that of the incidence graph is $\geq (m+1)/2$. While it happens that for $n = 1$ the CNF expression has an incidence graph with treewidth 1, as we increase n both CNF and DNF incidence graphs have large treewidth. On the other hand, $etwd(F_{mnp}) = 1$.*

[3]In most of the literature, the CNF form is used. We prefer DNF because it arises naturally in probabilistic databases.

2.4 Ordered Binary Decision Diagrams

OBDD were introduced by Bryant [5] and studied extensively in model checking and knowledge representation. A good survey can be found in [28]; we give here a quick overview. A *Binary Decision Diagram*, BDD, is a rooted DAG with two kinds of nodes. A *sink node* or *output node* is a node without any outgoing edges, which is labeled either 0 or 1. An *inner node* is labeled with a Boolean variable X_i and has two outgoing edges, labeled 0 and 1 respectively. Every node u uniquely defines a Boolean function as follows: $F_u = \texttt{false}$ and $F_u = \texttt{true}$ for a sink node labeled 0 or 1 respectively, and $F_u = \neg X_i \wedge F_{u_0} \vee X_i \wedge F_{u_1}$ for an inner node labeled with X_i and with successors u_0, u_1 respectively. The BDD represents a Boolean function $F \equiv F_u$ where u is the root of the BDD. An *Ordered* BDD, denoted OBDD, is such that there exists a total order Π on the set of variables, and on any path from the root to a sink every variable appears *at most once* and in the order Π (variables may be skipped). One also writes $OBDD_\Pi$, to emphasize that the order is Π. Given $1 \leq m \leq p \leq n$, denote $\Pi(m:p) = (\Pi(m), \ldots, \Pi(p))$; given $\bar{a} \in \{0,1\}^m$ we denote $F_{|\Pi(1:p)=\bar{a}}$ the Boolean function obtained from F by substituting the first m variables in Π with the values $\bar{a}$.

The *size* of an OBDD is the number of nodes, and the width at level m, $m \leq n$ is the number of distinct nodes labeled with the variable X_{m+1}. An upper bound on the width of OBDD_Π at level m is given by the number of subfunctions that result after substituting the first m variables, i.e. $|\{F_{|\Pi(1:m)=\bar{b}} \mid \bar{b} \in \{0,1\}^m\}|$. The width of an OBDD is the maximum width at any level. Obviously, the size of an OBDD of width w with n variables is $\leq n \cdot w$.

A *shared* BDD for a set of function $\bar{F} = \{F_1, F_2, \ldots, F_m\}$ is defined analogously, except it computes all the functions simultaneously. The sink nodes are labeled with $\{0,1\}^m$ and every node u represents m subfunctione $\{F_{u1}, F_{u2}, \ldots, F_{um}\}$, where F_{ui} can be obtained by applying the assignments until this node to F_i. From a shared OBDD for $\bar{F}$ one can construct an OBDD for any Boolean function over $\bar{F}$ of no larger size. This is often used in OBDD synthesis [28], where, instead of computing the OBDD of, say $F_1 \wedge F_2 \vee F_3$, one computes the shared OBDD of $\{F_1, F_2, F_3\}$.

2.5 Results on Expression-width and OBDD

We present here the results relating expression-width parameters to OBDD width/size.. We define four sets of Boolean functions:

DEFINITION 2.10.

$$RO = \{F \mid F \textit{ is read-once}\}$$
$$EPWD(k) = \{F \mid epwd(F) < k\}$$
$$ETWD(k) = \{F \mid etwd(F) < k\}$$
$$OBDD(w) = \{F \mid F \textit{ has an OBDD of width } \leq w\}$$

The size of any OBDD in $OBDD(w)$ is $\leq w \cdot n$; if $w = O(1)$ then the size of the OBDD is linear, but we also allow $w = w(n)$ to be a polynomial in n, and in that case the size is polynomial.

Our results are:

$$RO \subsetneq EPWD(O(1)) \equiv OBDD(O(1)) \subsetneq \subsetneq ETWD(O(1)) \subsetneq OBDD(n^{O(1)}) \quad (4)$$

We start by describing the containment results. Let F be a boolean function over $\bar{X} = X_1, X_2, \ldots, X_n$. Let $G = (V(G), E(G))$ be an expression DAG for F, and let $T = (V(T), E(T))$ be a tree-decomposition of G. We call T an *expression tree-decomposition* of F. Our first, and main result, shows that we can derive a "good" variable order Π from T, such that OBDD_Π has a width bounded by the width of T. We will describe now how to construct Π. We need to introduce some notations.

Recall that each node $Y \in V(T)$ is a set of nodes from $V(G)$, which, in turn, are labeled with a variable X_i or with an operator $\wedge, \vee, \neg$. Denote $Var(Y)$ the set of variables X_i occurring in Y: recall that, for any variable X_i, the set $\{Y \mid X_i \in Var(Y)\} \subseteq V(T)$ is connected. Denote $Var(V) = \bigcup_{Y \in V} Var(Y)$ for a subset $V \subseteq V(T)$. We say that Y *splits* the tree T into V_1, V_2 if $V_1 \cup V_2 = V(T)$, $V_1 \cap V_2 = \{Y\}$ and every path from $V_1 - \{Y\}$ to $V_2 - \{Y\}$ goes through Y.

Definition 2.11. *Let T be an expression tree decomposition of F, Π be permutation of the variables $\bar{X}$, and $m \leq n$. We say that $\Pi(1:m)$ is* compatible *with some tree node $Y \in V(T)$, if Y splits the tree into V_1, V_2 such $\Pi(1:m) \subseteq Var(V_1)$ and $\Pi(m+1:n) \subseteq Var(V_2)$.*

The following is the key technical lemma for our main result; we prove the lemma in Sect. 4.

Lemma 2.12. *If T is an expression path-decomposition of F, Π is a permutation of its variables, $\Pi(1:m)$ is compatible with Y for some $Y \in V(T)$, and $k = |Y|$, then:*

$$\left|\{F_{|\Pi(1:m)=\bar{b}} \mid \bar{b} \in \{0,1\}^m\}\right| \leq 2^{(k+1)*2^{k+1}} \quad (5)$$

Thus, if we have a tree decomposition of width $< k$ and $\Pi(1:m)$ is compatible with some node Y, then the width of OBDD_Π at depth m is bounded by the lemma. The bound in the lemma is almost tight, even for monotonic Boolean functions. To see this, let $m = 2^k$, consider $n = m + k$ variables $X_1, \ldots, X_m, Z_1, \ldots, Z_k$, and let $F = \bigvee_i \left(X_i \wedge \bigwedge_{j \in s_i} Z_j\right)$, where $s_1, \ldots, s_m$ represent all m subsets of $[k]$. Consider the path decomposition T with m nodes, $Y_i = \{X_i, \wedge_i, \vee\} \cup \bar{Z}$, where $\wedge_i$ denotes the i'th conjunct, and $\vee$ represents the root node in the expression DAG of F: the width is $k+2$. Let Π be the permutation $X_1, \ldots, X_m, Z_1, \ldots, Z_k$. Then $\Pi(1:m)$ is compatible with any node Y_i in the path T (by considering the split $V_1 = V(T)$ and $V_2 = \{Y_i\}$), yet the set $\{F_{|\Pi(1:m)=\bar{b}} \mid b \in \{0,1\}^m\}$ contains all monotonic Boolean function in the variables $\bar{Z}$: the number of such functions is the Dedekind number, $M(k)$, which is super-exponential, $M(k) \geq 2^{\binom{k}{k/2}}$. It is straightforward to extend the example to a non-monotonic Boolean function F, where the number of functions $F_{|\Pi(1:m)=\bar{b}}$ is equal to 2^{2^k}.

Ideally, we would like to find a permutation Π such that each of its prefix $\Pi(1:m)$ is compatible with some tree node Y: in that case we have a bound on the width of OBDD_Π. This is possible if T is a path; if it is a tree, we can still use the lemma and obtain a polynomial width.

A "good" permutation Π^R is defined by any *orientation* T^R of T, which is obtained by designating a node $R \in V(T)$ as the root node. Thus, T^R is a directed tree, and each $Y_i \in V(T)$ has a unique parent (except the root R) and a set of children $Y_{i_1}, Y_{i_2}, \ldots$. We denote T_i the subtree rooted at Y_i. Consider the following in-order traversal of the nodes $V(T)$, defined recursively, for each subtree. Assuming Y_i has c children, we order them such that $|Var(T_{i_1})| \geq |Var(T_{i_2})| \geq \ldots \geq |Var(T_{i_c})|$: then we traverse T_i in the order $T_{i_1}, Y_i, T_{i_2}, T_{i_3}, \ldots, T_{i_c}$, where each T_{i_j} is traversed recursively. This defines a total order of the nodes $V(T)$: $Y_1, Y_2, \ldots, Y_N$. For each i, consider the set of variables X_j first encountered at Y_i during this traversal: $FVar(Y_i) = Var(Y_i) - \bigcup_{j<i} Var(Y_j)$. Let Π_i be an arbitrary permutation of $FVar(Y_i)$. Then, we define the permutation $\Pi^R = \Pi_1, \ldots, \Pi_N$.

Corollary 2.13. *If T is an expression path decomposition of F of width $< k$, then for any node $R \in V(T)$, $OBDD_{\Pi^R}(F)$ has width at most $2^{(k+1)2^{k+1}}$.*

This implies $EPWD(k) \subseteq OBDD(2^{(k+1)2^{k+1}})$.

Proof. Notice that, when T is a path, then the tree traversal $Y_1, Y_2, \ldots, Y_N$, is simply a traversal of the path, from left to right or right to left (depending on the choice of R). Thus, Π^R lists the variables in the order in which they are first encountered on this path. For any prefix $\Pi^R(1:m)$, let $Y_j \in V(T)$ be the first node that contains the variable $\Pi^R(m)$, i.e. $\Pi^R(m) \in FVar(Y_j)$. We claim that $\Pi^R(1:m)$ is compatible with Y_j: indeed, Y_j splits the path into $V_1 = \{Y_1, \ldots, Y_{j-1}, Y_j\}$ and $V_2 = \{Y_j, Y_{j+1}, \ldots, Y_N\}$, all variables X_i in $\Pi^R(1:m)$ are in $Var(V_1)$, and all variables X_i in $\Pi^R(m+1, n)$ are in $Var(V_2)$. Thus, the width of the OBDD is given by the lemma. □

Corollary 2.14. *If T is an expression tree decomposition of width $< k$ of F, then for any node $R \in V(T)$, $OBDD_{\Pi^R}(F)$ has width at most $n^{2^{(k+1)2^{k+1}}}$.*

This implies $EPWD(k) \subseteq OBDD(n^{2^{(k+1)2^{k+1}}})$.

Proof. We use an inductive argument. Fix $Y_1, Y_2, \ldots, Y_N$ the in-order traversal of the tree T^R, denote T_i be the subtree rooted at Y_i, $\bar{X}_i = Var(T_i)$, and $n_i = |\bar{X}_i|$, for $i = 1, N$. Let $m = |\bigcup_{j<i} FVar(Y_j)|$ and denote $F_i = F_{|\Pi^R(1:m)=\bar{b}}$, for some $\bar{b} \in \{0,1\}^m$. That is, F_i denotes the result of substituting in F all variables encountered *before* reaching Y_i, with some values $\bar{b}$. Note that T^R is also an expression tree-decomposition of F_i, and that the permutation that T^R defines on the variables of F_i is precisely $\Pi^R(m+1:n)$. We prove, by induction on the node Y_i, that the width of $\text{OBDD}_{\Pi^R(m+1:n)}$ for F_i is $2^{\log n_i \cdot 2^{(k+1)2^{k+1}}}$; the corollary follows by applying this claim to the root node Y_i. Let M be any depth in this OBDD, $M = 1, n-m$. Let $Y_{i_1}, \ldots, Y_{i_c}$ be the children of Y_i, and let $T_{i_1}, T_{i_2}, \ldots, T_{i_c}$ be the subtrees rooted at these children. Consider where the variable $\Pi^R(m+M)$ is first encountered when traversing the tree T_i in the order $T_{i_1}, Y_i, T_{i_2}, \ldots, T_{i_c}$. If it is encountered in T_{i_1}, then the claim for F_i follows inductively from the claim about F_{i_1}, because $F_i = F_{i_1}$. If it is in Y_i, then Π^R_{m+M} is compatible with the node Y_i, because Y_i splits the tree into

$T_{i_1} \cup \{Y_i\}$ (which contains all variables in $\Pi^R(m+1, m+M)$) and the rest of the tree T (which contains all variables $\Pi^R(m+M+1, n)$), thus the claim follows by the lemma. If $\Pi^R(m+M)$ is first encountered in T_{i_j}, where $j > 1$, then let $\Pi^R(m+L)$ be the last variable *not* in T_{i_j} (thus $L < M$). Here, we first notice that the width of $\text{OBDD}_{\Pi^R(m+1:n)}$ for F_i at depth L is $\leq 2^{(k+1)*2^{k+1}}$: this follows from the lemma and the fact that $\Pi^R(m+1, m+L)$ is compatible with Y_i. Indeed, Y_i splits the tree into $T_{i_1} \cup \ldots \cup T_{i_{j-1}} \cup \{Y\}$ (which contains all of $\Pi^R(m+1, m+L)$) and the rest (which contains $\Pi^R(m+L+1, n)$). Thus, at depths L, $\text{OBDD}_{\Pi^R(m+1:n)}$ has width $\leq 2^{(k+1)*2^{k+1}}$: for each node at that level it has one copy of G_{i_j}. By induction, each such copy has a width $2^{\log n_{i_j} \cdot 2^{(k+1)2^{k+1}}}$. Now we use the fact that the subtrees T_{i_j} were ordered in decreasing number of variables. Since $j > 1$, its number of variables is $n_{i_j} \leq n_i/2$. It follows that, at depth M, $\text{OBDD}_{\Pi^R(m+1:n)}$ has width $\leq 2^{(k+1)*2^{k+1}} \times 2^{\log n_{i_j} \cdot 2^{(k+1)2^{k+1}}} \leq 2^{\log n_i \cdot 2^{(k+1)2^{k+1}}}$, proving our inductive claim about F_i. $\square$

Next, we state the surprising converse for Corollary 2.13.

THEOREM 2.15. *If there exists an OBDD for F with width w, then there exists an expression DAG G representing F s.t. $pwd(G) \leq 5w$.*

This implies $OBDD(w) \subseteq EPWD(5w+1)$.

Finally, we connect to *RO*. The following is folklore:

PROPOSITION 2.16. *Every read-once Boolean function has an OBDD of width ≤ 1. Thus, $RO \subseteq OBDD(1)$.*

This can be shown by induction on the size of the expression. The OBDD for $E_1 \wedge E_2$ consists of a copy of the OBDDs for E_1 and E_2, with the 1-labeled sink node of the former replaced by the root node of the latter[4]; similarly for $E_1 \vee E_2$. Thus:

COROLLARY 2.17. *$RO \subseteq EPWD(6)$*

This completes our description of the containment results in Eq. (4). The separation results are as follows. The first separation is given by the lineage of a query in UCQ, and the last separation is given by the lineage of a query in $UCQ^{\neq}$; they will both be discussed in Sect. 3. We show here the second separation. For that, we define the Boolean functions bt_m over 2^m variables, where m is even:

$$\begin{aligned} bt_m(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4) &= (bt_{m-2}(\bar{x}_1) \oplus bt_{m-2}(\bar{x}_2)) \wedge \\ &\quad (bt_{m-2}(\bar{x}_3) \oplus bt_{m-2}(\bar{x}_4)) \qquad (6) \\ bt_0(x) &= x \end{aligned}$$

where $\bar{x}_i, i = 1..4$ are variable vectors of size 2^{m-2} and $\oplus$ is the XOR-operator. This is a read-once expression in the extended grammar Eq. (3): it is *not* a read-once expression using our regular grammar Eq. (2). Hence, $etwd_*(bt_m) = 1$, and, by Prop. 2.5, we have $etwd(bt_m) \leq 3$, hence $bt_m \in ETWD(4)$. On the other hand we show the following, which separates $OBDD(O(1)) \subsetneq ETWD(O(1))$:

THEOREM 2.18. *Any OBDD for bt_m has width $\geq \frac{2^{m/2}}{2}$. Thus, $bt_m \notin OBDD(w)$ for any constant w.*

[4]Notice that number of subfunctions $F_{|\Pi(1:m)=\bar{b}}$ of a read-once expression is, in general, unbounded.

3. RESULTS ON QUERY COMPILATION

In this section we discuss applications to query compilation (the right diagram of fig. 2), and also use them to derive separation results.

3.1 Background: UCQ and $UCQ^{\neq}$

A conjunctive query, $q = R_1(\bar{x}_1) \wedge R_2(\bar{x}_2) \wedge \ldots \wedge R_m(\bar{x}_m)$ is a conjunction of relational atoms $R_i(\bar{x}_i)$, where $\bar{x}_i$ consists of variables and constants, and R_i are symbols from a fixed vocabulary. An inequality predicate over q is of the form $x \neq y$, or $x \neq a$, where x, y are variables and a is a constant. A Union of Conjunctive Query with inequalities ($UCQ^{\neq}$) is defined as $Q = \bigvee_{i=1}^{k} (q_i \wedge p_i)$, where $q_1 \ldots q_k$ are conjunctive queries and p_i is a conjunction of pairwise inequality predicates over q_i. An example is $R(x), S(x,y), x \neq y \vee R(x), T(y)$, where we have used comma for $\wedge$, a convention we adopt in the rest of the paper as well. A *Union of Conjunctive Query* (UCQ) is a query without inequalities. All queries in this paper are Boolean queries.

Let D be a database instance. Denote X_t a distinct Boolean variable for each tuple $t \in D$. Let Q be a $UCQ^{\neq}$. The *lineage* of Q on D is a Boolean expression $F(Q, D)$ over $\bar{X}$ s.t. for any $D' \subseteq D$, $D' \models Q$ iff the assignment $X_t = \texttt{true}$, if $t \in D'$ and $\texttt{false}$ otherwise, satisfies $F(Q, D)$. Figures 1, 1 have examples where lineage is represented as an expression DAG. Green at al. [14] describe a general algorithm for computing the semiring annotation of a query output, by using an relational algebra plan for the query: this can be used to compute the lineage $F(Q, D)$, and also to derive an expression DAG for it.

In this paper, we are only interested in the *data complexity*, hence we assume query to be fixed, and the database to be variable. Thus, each query defines a set of Boolean functions, and we denote:

DEFINITION 3.1. *For any $C \in \{UCQ, UCQ^{\neq}\}$, define*

$$\begin{aligned} C(RO) &= \{Q \in C \mid \forall D.F(Q,D) \in RO\} \\ C(EPWD) &= \{Q \in C \mid \exists k \forall D.F(Q,D) \in EPWD(k)\} \\ C(ETWD) &= \{Q \in C \mid \exists k \forall D.F(Q,D) \in ETWD(k)\} \\ C(OBDD) &= \{Q \in C \mid \exists w \forall D.F(Q,D) \in OBDD(w)\} \\ C(OBDD_{poly}) &= \{Q \in C \mid \exists k \forall D.F(Q,D) \in OBDD(|D|^k)\} \end{aligned}$$

We assume our queries to be *ranked*, [8, 26], which means that the query has no constants (and, hence, no predicates of the form $x \neq a$), and there exists a global order $\prec$ on the variables such that, whenever x, y occur in a common relational atom and x precedes y, then $x \prec y$. Every query is equivalent to (meaning that it has the same lineage as) a ranked query over some different relational vocabulary; for example, $R(x,y), R(y,x)$ is equivalent to $R_1(x,y), R_2(x,y) \vee R_3(z)$, where $R_1 = \sigma_{x<y}(R)$, $R_2 = \Pi_{yx}(\sigma_{x>y}(R))$, and $R_3 = \Pi_x(\sigma_{x=y}(R))$ form a partition on R; we refer to [26] for details.

3.2 Background: Inversion-Free Queries, IF

Given a conjunctive query q, its Gaifman graph is a graph with nodes as the variables in query and two variables are connected if they are present together in some atom in the query. A *component* c is a conjunctive query whose Gaifman graph is *connected*. Hence, every conjunctive query q can be expressed as $q = c_1 \wedge c_2 \wedge \ldots \wedge c_k$, where each c_i is a component, and for all $i \neq j$, c_i, c_j do not share common variables.

We denote the set of components $C(q) = \{c_1, c_2, \ldots, c_k\}$. Given a $UCQ^{\neq}$ query $Q = \bigvee_i (q_i \wedge p_i)$, we define its components as $C(Q) = \bigcup_{i=1} C(q_i)$.

DEFINITION 3.2. *Let c be a component. A variable x is a* root variable *if it occurs in all atoms of c.*

Let $\bar{c} = \{c_1, c_2, \ldots, c_m\}$ be a set of components. A set of variables $\bar{x} = \{x_1, x_2, \ldots, x_m\}$ is a separator *if for each relational symbol R there exists a number i_R such that for all $j = 1, m$, every relational atom in c_j with symbol R has the variable x_j on position i_R. In particular, x_j is a root variable in c_j.*

EXAMPLE 3.3. *The query $R(x), S(x, y)$ has root variable x; the query $R(x), S(x, y), T(y)$ has no root variables.*

The set of components $\{[R(x_1), S(x_1, y_1)], [S(x_2, y_2), T(x_2)]\}$ has separator x_1, x_2. Indeed, $i_R = i_S = i_T = 1$; note that both S-atoms have the separator variable on position 1. On the other hand, $\{[R(x_1), S(x_1, y_1)], [S(x_2, y_2), T(y_2)]\}$ has no separators: the set x_1, y_2 is not a separator because we cannot take either $i_S = 1$ or $i_S = 2$: x_1 occurs on the first position in $S(x_1, y_1)$, while y_2 occurs on the second position in $S(x_2, y_2)$.

A ground atom is an atom without variables: since the query is ranked, this must be a nullary relation symbol $R()$ (a ground tuple with constants, like $R(a, b)$, is assimilated with a nullary symbol while ranking the query [26]). A ground atom is a component by itself. If $\bar{c}$ is a set of components, denote $\bar{c}^+ \subseteq \bar{c}$, the subset of components that have at least one variable, i.e., we remove ground atoms. Let $\bar{x}$ be a separator for $\bar{c}^+$. Define a new vocabulary where each relation R has the arity decreased by one, and is obtained by removing the attribute i_R; denote $c_{i,-\bar{x}}$ the conjunctive query obtained from the component c_i by removing from each atom R the attribute i_R. Notice that $c_{i,-\bar{x}}$ is not necessarily connected. Let $\bar{c}^+_{-\bar{x}} = \bigcup_i C(c_{i,-\bar{x}})$, be the new set of components, where c_i ranges over $\bar{c}^+$.

DEFINITION 3.4. *A set of components $\bar{c}$ is* consistently hierarchical *if either $\bar{c}^+$ is empty or has a separator $\bar{x}$ s.t. $\bar{c}^+_{-\bar{x}}$ is consistently hierarchical.*

DEFINITION 3.5. *$IF^{\neq}$ is the set of all $UCQ^{\neq}$ queries Q, s.t. $C(Q)$ is consistently hierarchical.*

We call queries in $IF^{\neq}$ inversion-free. Denote IF the set of inversion-free queries that do not use $\neq$.

EXAMPLE 3.6. *Consider the query shown in Fig. 1, $q_{rsst} = R(x_1), S(x_1, y_1), T(x_2), S(x_2, y_2)$. We prove that it is inversion-free. $C(q_{rsst}) = \bar{c} = \{[R(x_1), S(x_1, y_1)], [S(x_2, y_2), T(x_2)]\}$ has separator x_1, x_2. Then $\bar{c}^+_{-\bar{x}} = \{[R()], [S(y_1)], [S(y_2)], [T()]\}$. After removing the ground atoms, we obtain $\{[S(y_1)], [S(y_2)]\} \equiv \{S(y_1)\}$ which has separator y_1, proving that $\bar{c}$ is consistently hierarchical. Thus, $q_{rsst} \in IF$.*

3.3 Results for UCQ

The following result is known from [18]:

THEOREM 3.7. *[18] If $Q \in IF$, then for any database D, $F(Q, D)$ admits an OBDD of width 2^g, where g is the number of atoms in Q. Furthermore, if $Q \in UCQ - IF$, then there exists a database D over which no OBDD for $F(Q, D)$ has size polynomial in D.*

From here we derive immediately the collapse of the following classes:

COROLLARY 3.8. *The following hold: $UCQ(EPWD) = UCQ(ETWD) = UCQ(OBDD) = UCQ(OBDD_{poly}) = IF$*

For example, for any input database, the lineage of q_{rsst} has an OBDD of width 15 and, from here, it follows from Theorem 2.15 that it has an expression DAG of path width ≤ 80. Notice that this is not obvious at all from examining the lineage expression for q_{rsst} in Fig. 1, since the "natural" lineage has an unbounded tree-width: we need the detour through OBDD to obtain a bounded path-width expression.

It is also known from [18] that $q_{rsst} \notin UCQ(RO)$, thus $UCQ(RO) \subsetneq UCQ(OBDD)$, proving the first separation in Eq. (4). As discussed in Sect. 1, $q_{rs} \in UCQ(RO)$.

3.4 Results for $UCQ^{\neq}$

We present here our main result on query compilation. All proofs are in Sect. 5.

THEOREM 3.9. *$IF^{\neq} = UCQ^{\neq}(OBDD_{poly})$.*

Unlike for IF, the OBDD are no longer of constant width though. Therefore, queries in $IF^{\neq}$ are excellent candidates for separating constant-width OBDD from polynomial-size OBDD. We do this with the following simple query $q_{nr} = R(x_1, y_1), R(x_2, y_2), x_1 \neq x_2, y_1 \neq y_2$. Since q_{nr} is inversion-free, we have $q_{nr} \in UCQ^{\neq}(OBDD_{\text{poly}})$. We show:

THEOREM 3.10. *For any c, there exists a database D, s.t. $etwd(F(q_{nr}, D)) > c$. This implies that $UCQ^{\neq}(ETWD) \subsetneq UCQ^{\neq}(OBDD_{poly})$.*

The theorem immediately implies the last separation result in Eq. (4), $ETWD(O(1)) \subsetneq OBDD(n^{O(1)})$. It turns out that there exists a matching upper bound for q_{nr}: for any database D, the lineage of q_{nr} on D has an OBDD of width linear in the number of tuples of D.

Since $UCQ^{\neq}(OBDD_{\text{poly}})$ has a syntactic characterization, a natural question arises if there exists a syntactic characterization for $UCQ^{\neq}(ETWD)$. We define a language, $H^{\neq}$, as a fragment of $IF^{\neq}$, and prove that $H^{\neq} \subseteq UCQ(EPWD)$, which implies that membership in $H^{\neq}$ is a sufficient condition for $UCQ^{\neq}(ETWD)$.

DEFINITION 3.11. *Given a set of components $\bar{c} = c_1 \ldots c_k$ and a set of inequality predicates P, we say $(\bar{c}, P)$ is consistently hierarchical if $\bar{c}^+$ is empty or there exists a separator $\bar{x}$ of $\bar{c}^+$ s.t. (i) for every predicate $z \neq y$ in P s.t. $z \notin \bar{x}$ and $y \notin \bar{x}$, and any component c_i : if z appears in c_i, then so does y and vice-versa, and (ii) let $P_{-\bar{x}}$ be the set of predicates from P that do not involve any variable from $\bar{x}$, then $(\bar{c}^+_{-\bar{x}}, P_{-\bar{x}})$ is consistently hierarchical.*

Define $H^{\neq}$, a subset of $IF^{\neq}$, as the set of all queries of the form $Q = \bigvee_{i=1}^{k} (q_i \wedge p_i)$ s.t. $(\bigcup_i C(q_i), \bigcup_i p_i)$ is consistently hierarchical. Clearly, $IF \subset H^{\neq}$ by its definition. We have the result

THEOREM 3.12. *$H^{\neq} \subseteq UCQ(EPWD)$*

EXAMPLE 3.13. *$R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2), x_1 \neq x_2$ is in $H^{\neq}$ because it only compares two separator variables. For another example, $R(x, y), S(x, z), y \neq z$ is also in $H^{\neq}$:*

the inequality is from within the same component, and, after removing the separator variable x, the inequality $y \neq z$ is between separator variables.

$R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2), x_1 \neq x_2, y_1 \neq y_2$ is not in $H^{\neq}$ since y_1, y_2 belong to two different components.

The query q_{nr} is not in $H^{\neq}$. There are two components, $R(x_1, y_1)$ and $R(x_2, y_2)$ and we have two choices of separator variables: either x_1, x_2 , or y_1, y_2. But with either choice, one of the inequality conditions violates the condition of $H^{\neq}$.

We conjecture that $UCQ^{\neq}(EPWD) = UCQ^{\neq}(ETWD)$.

4. PROOFS ON OBDD VS TREEWIDTH

PROOF OF LEMMA 2.12. We call the variables from $\Pi(1 : m)$ *set* and the rest as *unset*. Let $Y = \{w_1, w_2, \ldots, w_k\}$ where $\bar{w}$ are vertices of the expression DAG G for F. Now we associate a formula g_u to each node u in G as follows : for all w_j, it is a new variable f_j; else wither $u = v \otimes w$, then $g_u = g_v \otimes g_w$, or u is a leaf X_i, then $g_u = X_i$. Suppose $F = F_1 \otimes F_2$, then F_1, F_2 can be defined as formulae over $\bar{f}$ and some subset of variables $\bar{X}_1, \bar{X}_2$ respectively from $\bar{X}$. We claim that either every variable in $\bar{X}_1$ is *set* or they are all *unset* and same for $\bar{X}_2$. Suppose variables X_{i1}, X_{i2} are in $\bar{X}_1$ and $X_{i1} \in Var(V_1)$, $X_{i2} \in Var(V_2)$, where V_1, V_2 are as in Def. 2.11. But both X_{i1}, X_{i2} must be connected to F_1 by some path since F_1 depends on these variables, hence F_1 must belong in Y. But then $F_1 = f_j$ for some j and it shouldn't depend on any X_i; a contradiction. We could argue similarly for F_2.

Now we will bound the number of possible values F_1, F_2 can take as variables from $\Pi[1 : m]$ are set to 0/1. We will prove for F_1 and symmetrically the same can be shown for F_2. Suppose $\bar{X}_1$ are all set. Then F_1 is a Boolean function over k variables $\bar{f}$. There are only 2^{2^k} possible Boolean functions over k variables. Now, each variable f_i stands for the function represented at the gate w_i. Let assume each f_i take at most N possible values too. Then, we have a bound of $2^{2^k} N^k$ on the number of possible values of F_1. Similarly, if all $\bar{X}_1$ were unset, the bound would be N^k. We will now bound N.

For any w_j which has l nodes from $\bar{w}$ as its descendants, we claim that f_j takes at most 2^{2^l} values. If w_j doesn't have any nodes from $\bar{w}$ as its descendant, then N is either 1 or $2 = 2^{2^0}$. Suppose this holds for all nodes below w_j. We use the same argument as above again. Consider the set of nodes $\bar{u} = \{u_1, u_2, \ldots, u_t\}$ from $\bar{w}$ directly below w_j i.e. there exists no other node w_i which is a descendant of w_j and the ancestor of the said node. Then $f_j = f_{j1} \otimes f_{j2}$, where f_{j1}, f_{j2} are boolean formulae over $\bar{u}$ and other nodes, all of which must be *set* or *unset*. Hence the number of possible values of f_{j1}, f_{j2} is at most $2^{2^l} \prod_i 2^{2^{l_i}}$, where l_i is the number of nodes from $\bar{w}$ below u_i. Since $l + \sum_i l_i \leq k-1$, we get $N \leq 2^{2^k}$. Hence both F_1, F_2 take at most $2^{(k+1)2^k}$ values, hence the number of possible valuations of F can't be more than $2^{2 \cdot (k+1)2^k} = 2^{(k+1)2^{k+1}}$ □

PROOF OF TH. 2.15. We construct an expression G for F s.t. $pwd(G) = 5w$. We do this using the concept of a *shared expression DAG* : Given formulas $f_1, f_2, \ldots, f_w$, a shared expression DAG is an expression DAG which represents all the $\bar{f}$, i.e., it has w root or output nodes and the formula obtained by evaluating the expression below these nodes corresponds exactly to $\bar{f}$. The construction follows from the following more general lemma.

LEMMA 4.1. *If $\bar{f} = f_1, f_2, \ldots, f_w$ have a shared OBDD with width w, then there exists a shared expression for them having path width $5w$, s.t. all root nodes $\bar{f}$ occur on a leaf of the path decomposition.*

PROOF. We prove by induction on the number of variables n. Let the first variable in the variable order of OBDD be X_1 and denote the formulae at the first level by $g_1, g_2, \ldots$,g_w. Then every f_i can be written as $\neg X_1 \wedge f_j \vee X_1 \wedge f_k$ for some j, k. Denote the nodes corresponding to new $\wedge, \neg$ operators by $\overline{op}$. Now by induction hypothesis, $\bar{g}$ have a path-decomposition with width $5w$ one of whose leaves contains $\bar{g}$. We connect that leaf to a new node which contains $\bar{g}, \bar{f}, X_1, \overline{op}$. The resulting path-decomposition of $\bar{f}$ has width $5w$. □

PROOF OF TH. 2.18. We will prove that an OBDD of $\{bt_m, \neg bt_m\}$ has width $\geq 2^{m/2}$. Since the width of a function and its negation are the same, we get that the width of bt_m is at least $\frac{2^{m/2}}{2}$. We will proceed by induction. In the base case($m = 0$), width of $\{x, \neg x\}$ is $2 \geq 2^0/2$. Let $bt_m = f = (f_1 \oplus f_2) \wedge (f_3 \oplus f_4)$, where $f_i, i = 1 \ldots 4$ is bt_{m-2}, and hence width of $\{f_i, \neg f_i\}$ is at least $2^{m/2-1}$.

Consider the first level where all variables of one of the $\bar{f}$ have been *set* to 0/1. : say f_1. We first consider the case that X, a variable of one of f_2, f_3, f_4 has been set before this level. By induction hypothesis(IH), we know that there exists a level, where width of f_1 is $w = 2^{m/2-1}/2$. Let $\{u_1, u_2, \ldots, u_w\}$ be the corresponding w subformulae of f_1. We have two cases depending on whether X is a variable of f_2 or f_3, f_4.

Case I : X is a variable of f_2. Let g_1, g_2 be two distinct subformulae of f_2 at this level. Pick a subformula h of $f_3 \oplus f_4$ from this level, s.t. $h \neq \texttt{false}$. We define 4 sets of formula at this level :

$$S_i = \{(u_p \oplus g_i) \wedge h) \mid 1 \leq p \leq w\}, i = 1, 2$$
$$T_i = \{\neg\,((u_p \oplus g_i) \wedge h) \mid 1 \leq p \leq w\}, i = 1, 2$$

So S_1, S_2 contain the subformula from f and T_1, T_2 from $\neg f$. First, we claim that no formula from $\bar{S}$ can equal any from $\bar{T}$. This is simple to see since setting $h = 0$ sets any formula in $\bar{S}$ to 0 while in $\bar{T}$ to 1. Hence they can't be equal. Now, we compare the functions in S_1, S_2. Clearly $(u_i \oplus g_1) \wedge h \neq (u_i \oplus g_2) \wedge h$, since setting g_1, g_2 to 0,1 gives two different functions $u_i \wedge h, \neg u_i \wedge h$. Now, suppose $(u_i \oplus g_1) \wedge h = (u_j \oplus g_2) \wedge h$, then if we set $g_1 = g_2$ to 0 or 1, we get $u_i = u_j$. We can do this because one of g_i, g_j must not be 0/1, as not all variables of f_2 have been set. Hence we get functions in S_1, S_2 are also distinct and similarly the same holds for $\bar{T}$. Therefore, total width $= 4w = 4 \cdot 2^{m/2-1}/2 = 2^{m/2}$.

Case II : X is a variables of f_3 or f_4. This case is also similar and easy to verify as all 4 classes lead to different formulae.

So, now we can assume that all variables of f_1 are set and no other variables have been. Again, we need to consider two cases depending on whether we start with f_2 or f_3, f_4 next. Note that, by the same argument as above, we will be setting all variables of one of them.

Case III : Suppose we set all variables of f_2 next. Let $h = f_3 \oplus f_4$. After exhausting variables of f_1, we have 4 different formulae in the OBDD of $\{f, \neg f\}$ which we group as : $(f_2 \wedge h, \neg f_2 \wedge h)$ and $(\neg(f_2 \wedge h), \neg(\neg f_2 \wedge h))$. Now lets consider a level where the width of an OBDD for $\{f_2, \neg f_2\}$ is $w = 2^{m/2-1}$. Then clearly we get 2 sets of subformulae of cardinality w each from the two groups mentioned above. And as argued in the previous cases, the two cannot have any formula in common, since setting $h = 0$ leads to different value in two cases. Hence, we get width at this level $= 2w = 2 * 2^{m/2-1} = 2^{m/2}$.

Case IV : In this case we set variables of f_3 next. Its easy to see that in this case, all 4 resulting formulae must lead to distinct subformulae. And the width for each is at least $2^{m/2-1}/2$. Hence width is at least $4 \cdot 2^{m/2-1}/2 = 2^{m/2}$.

This exhausts all possible cases, hence proved. □

5. PROOFS ON OBDD SIZE OF $IF^{\neq}$

First, we describe how to get rid of all predicates of the form $x \neq a$, where a is a constant. This can be accomplished by rewriting the query and suitably changing the vocabulary just as we did for removing constants. We just illustrate this with an example :
$R(x_1), S(x_1, y_1), S(x_2, y_2), R(x_2), x_1 \neq x_2 \wedge x_2 \neq a.$
We can rewrite this to :
$R^{-a}(x_1), S^{-a}(x_1, y_1), S^{a}(y_2), R^{a}(), x_1 \neq x_2 \vee$
$R^{-a}(x_1), S^{-a}(x_1, y_1), S^{-a}(x_2, y_2), R^{-a}(x_2), x_1 \neq x_2.$
The resulting query is still inversion-free with the same inequalities.

Now, we show that all queries in $H^{\neq}$ have an OBDD of width $O(1)$.

Proof Of Th. 3.12. Let $P = \{p_1, p_2, \ldots, p_k\}$ be the set of pairwise inequality predicates in Q and $C(Q) = \bar{c} = c_1, c_2, \ldots, c_l$ be the set of components. For any $s_1 \subseteq [k], s_2 \subseteq [l]$, let $q_{s_1,s_2} = \bigwedge_{i \in s_1} c_i \ , \ \bigwedge_{j \in s_2} p_j$. We will construct a shared OBDD for all 2^{k+l} queries q_{s_1,s_2}. One can derive the OBDD for Q from this shared OBDD.

Let $\bar{x} = x_1, x_2, \ldots, x_l$ be a separator for $\bar{c}$. Let the active domain of $\bar{x}$ be $\{a_1, a_2, \ldots, a_n\}$. Assume inductively that the width of each of $q_{s_1,s_2}[a_i/\bar{x}]$ depends only on the query. Now, note that since we have no inequality predicates between non-root variables, the OBDD for $\bar{x} = a_i$ and $\bar{x} = a_j$ can be constructed independently for $i \neq j$. Suppose we have constructed the shared OBDD for all $q^{i-1}_{s_1,s_2} = q_{s_1,s_2}, \bar{x} \in \{a_1, a_2, \ldots, a_{i-1}\}$. We show how to extend it to get the shared OBDD of all $q^{i}_{s_1,s_2}$. Note that $q^{i}_{s_1,s_2}$ is true iff there exists some $r_1 \subseteq s_1, r_2 \subseteq s_2$, s.t. $q^{i-1}_{r_1,r_2}$ is true and for all $j \in s_1 - r_1$, $c_j[a_i/\bar{x}]$ is true, and for any predicate $x_o \neq x_p$ in $s_2 - r_2$, $c_o[a_i/\bar{x}]$ is true and $p \in r_1$ or vice-versa $c_p[a_i/\bar{x}]$ is true and $p \in r_1$. One can hence construct the shared OBDD for all $q^{i}_{s_1,s_2}$ by just following the above logic. The width is at most $2^{2^{k+l}} = O(1)$. □

Now we show that all queries in $IF^{\neq}$ have an OBDD of polynomial size. The converse is a straightforward generalization of the proof from [18] and we discuss that in the Appendix.

Proof Of Th. 3.9. Let $q^{\neq} = q, P$, where q is an inversion-free conjunctive query and P is a a conjunction of inequality predicates. We claim that if for a variable order Π, $OBDD_{\Pi}(q)$ has polynomial size, then so does $OBDD_{\Pi}(q^{\neq})$. By the classical result on synthesis of OBDD, we have that if $OBDD_{\Pi}(F_1)$ has size s_1 and $OBDD_{\Pi}(F_2)$ is of size s_2, then $OBDD_{\Pi}(F_1 \otimes F_2)$ is of size at most $s_1 s_2$, where $\otimes$ is any boolean connective viz. OR,XOR. So if $Q = \bigvee_{i=1}^{k} (q_i \wedge p_i)$ is inversion-free : we know there exists Π under which all q_i have polynomial size OBDD, and hence by our claim so do q_i, p_i and by the synthesis result Q has a polynomial size OBDD. So it suffices to prove the claim that $q^{\neq}$ has a polynomial size OBDD.

Given any tuple t, we define $P(t)$ as the formulae obtained by setting variables associated with tuple t in P. For example if P is $x \neq y$ over $R(x), S(x, y)$, then $P(R(1)) = y \neq 1$. Given a vector of tuples $\bar{t}$ and boolean assignments $\bar{a}$ on them, we similarly define $P[\bar{a}/\bar{t}] = \bigvee_{a_i = \texttt{true}} P(t_i)$.

We already know that there exists an OBDD of constant width for q. Now consider any branch of tuples $\bar{t}$ with assignment $\bar{a}$ on this OBDD. Given a database D, if the subformulae/lineage of q represented by this branch was f, then for $q^{\neq}$, it is $F(q^{\neq}, D - \bar{t}) \vee (f, P[\bar{a}/\bar{t}])$, where $F(q^{\neq}, D - \bar{t})$ denotes the lineage obtained by evaluating $q^{\neq}$ over D without the tuples $\bar{t}$. Since our OBDD has constant width, for most of the assignments $\bar{a}$ on $\bar{t}$, the resulting formula f will be the same. We will further show that as we iterate over exponentially many assignments $\bar{a}$, the number of distinct predicates $P[\bar{a}/\bar{t}]$ generated is only polynomial. Hence, since the width of the original OBDD was constant, the new width only increases polynomially.

Suppose $P = \bigwedge_{j=1}^{k} x_{i1} \neq y_{j1}$. Suppose there are m variables in $\bar{x} = x_1, x_2, \ldots, x_m$ and w.l.o.g. assume the tuples that we have seen only influence the variables $\bar{y}$. Consider $DR = \neg P[\bar{a}/\bar{t}]$. It looks like :

$$\begin{aligned}(x_{i1} = a_{1,j1} \vee x_{i2} = a_{1,j2} \vee \ldots x_{ik} = a_{1,jk}) \wedge \\ (x_{i1} = a_{2,j1} \vee x_{i2} = a_{2,j2} \vee \ldots x_{ik} = a_{2,jk}) \wedge \\ \ldots\ldots\ldots \wedge \\ (x_{i1} = a_{n,j1} \vee x_{i2} = a_{n,j2} \vee \ldots x_{ik} = a_{n,jk})\end{aligned}$$

where n tuples were observed to be true. At a quick glance it would seem that the number of minterms in DR could be nk. But many of them are actually inconsistent, since we can't have two terms like $x_1 = a_p$ and $x_2 = a_q$ in the same minterm. In particular, every minterm is of length at most m. We show that there are at most 2^{mk} minterms in DR. Now, there are at most $(1 + |D|)^m$ possible minterms of size less than m and hence we get number of possible DR is bounded by $(1 + |D|)^{m2^{mk}}$. We prove the claim next.

Lemma 5.1. *The number of minterms in DR is at most* 2^{mk}.

Proof. We prove by induction on k. Consider the following minterm T of DR : $(x_1 = a_1 \wedge x_2 = a_2 \wedge \ldots \wedge x_m = a_m)$. Each clause in DR must contain one of $x_i = a_i$, hence we can write DR as :

$$\begin{aligned}(x_1 = a_1 \vee C_{11}) \wedge (x_1 = a_1 \vee C_{12}) \wedge \ldots \wedge \\ (x_2 = a_2 \vee C_{21}) \wedge (x_2 = a_2 \vee C_{22}) \wedge \ldots \wedge \\ \ldots \wedge \\ (x_m = a_m \vee C_{m1}) \wedge (x_m = a_m \vee C_{m2}) \wedge \ldots \qquad (7)\end{aligned}$$

Consider an arbitrary minterm T' of DR. We write T' as $T' = T_0 \wedge T_1$, where T_0 contains all inequalities$(x_i = a_i)$ that are common between T' and T, and T_1 contains the others,

i.e. not occurring in T. We will count the minterms T' by grouping by T_0 : there are 2^m possibilities for T_0, and for each of them we will count the number of distinct minterms T_1.

We know that $T' \Rightarrow DR$, hence if we substitute $(x_i = a_i)$ to be true for every $x_i = a_i$ term in T_0, we obtain $T_1 \Rightarrow DR'$, where DR' is obtained from Eq. (7) by removing all rows with index i s.t. $x_i = a_i$ is in T_0. Now we know that T_1 doesn't contain any other minterm of the form $x_j = a_j$ that is left in DR', hence we can remove them from DR to get DR'' and the implication $T_1 \Rightarrow DR''$ would still hold. So we get that T_1 is a minterm of an expression, DR'', which by induction hypothesis has at most $2^{m(k-1)}$ minterms. Therefore number of possible choices for T_1 are at most $2^{m(k-1)}$. Since $T' = T_0 \wedge T_1$, we get number of possibilities for T' are at most $2^m 2^{m(k-1)} = 2^{mk}$. Hence proved. □

Proof Of Theorem 3.10. We will first study the OBDD for q_{nr}. In particular, we will show that it has an OBDD of linear size and show a matching lower bound too. Then we will use the lower bound proof to prove the theorem.

Consider $R = \{(i, j) \mid 1 \leq i, j \leq n\}$ and the order Π on tuples which just sorts them in natural increasing order. We will show that over this database, the OBDD with this order has width $O(n)$. Note that the OBDD for Q on any other database where the domain size of attributes is n or less can be obtained by just setting some tuples in this OBDD to 0 ; the resulting OBDD would still have width $O(n)$. This proves the upper bound. For the lower bound, we'll show a level where the width has to be $n/2$, regardless of the order used. We associate a random variable r_{ij} with the tuple $R(i, j)$ to represent lineage of Q, which we call ϕ.

Note that

$$\phi_{|r_{pq}=1} = \left(\bigvee_{i \neq p} r_{iq} \wedge \bigvee_{j \neq q} r_{pj} \right) \vee \bigvee_{i \neq p, j \neq q} r_{i,j} \qquad (8)$$

After setting any subset of the rest of the variables to 0/1, there are only $O(n)$ possible formula from $\phi_{|r_{pq}=1}$. Now at any level, there is only one branch where all variables are set to 0. All other branches have at least one variables set to 1, and as we saw there are only $O(n)$ possibilities for the formula that result from these assignments. Hence the width of the OBDD is $O(n)$.

Now, for the lower bound consider the $n/2$ level. We consider all assignments where exactly one of the variables is set to 1. There are $n/2$ such assignments. For each of these assignments, by inspection of Eq. (8), we see that we get a different formula that still *depends* on every variable not yet set. Hence the width is at least $n/2$.

Now we prove the theorem. Consider the same data instance of R as above. Note that the argument for the lower bound proof above actually proves that the width at any level l, s.t., $cn \leq l \leq n-1$, for any $c < 1$, is l. Suppose there is an expression DAG for ϕ, the lineage of q_{nr}, with constant width tree-decomposition T^R. For the purpose of this proof we will assume the tree-decomposition is *normal*[13]. This means T is binary and if $Y_i \in V(T)$ has two children Y_j, Y_k then $Y_i = Y_j = Y_k$ and if Y_i has only one child Y_j, then $|Y_i - Y_j| = 1$. Now suppose we start with R as in the construction from Corollary 2.14. Consider the neighbor Y_1 : the number of $\bar{r}$ variables in descendants of Y_1 is at least $n^2/2$. We similarly pick a child of Y_1 with at least $n^2/4$ $\bar{r}$ variables in its descendant. We can keep iterating and each time the number of variables in the descendants of the chosen node can decrease by at most half. Hence at one point, we must reach a node Y s.t. the number of $\bar{r}$ variables in its descendants is between $n/4$ and $n/2$. Then, after setting all variables in the descendants of Y, we must get constant number of subformulae according to Lemma 2.12, while our lower bound proof suggests the number of subformulae is at least $n/4$; a contradiction. □

6. CONCLUSION AND FUTURE WORK

We have presented a new notion of treewidth for Boolean formula, called expression treewidth, that captures many of the tractable cases known in probabilistic databases. We have shown that bounded expression treewidth implies polynomial size OBDD. Furthermore, bounded expression pathwidth is equivalent to constant-width OBDD. Since both parameters : treewidth,OBDD, are widely used in areas like SAT, CSP, Verification, Graphical Models, etc., we think these connections can have wide-ranging applications.

The computability of this parameter though is still an open problem and presents interesting avenues for future research. Also, with all these structural parameters, we can only capture the set of queries which have an OBDD. The set of queries for which model counting is possible, is known to be much larger than this set [18] and there are other compilation languages, like FBDD, d-DNNF, which can solve more queries than OBDD. This motivates the need to find a structural parameter that could at least capture FBDD.

We would also like to investigate other width-parameters beyond treewidth which are more general and can solve problems unsolvable by using tree-decompositions. Clique-width are known to be another parameter under which model counting is tractable. In fact [11] showed that if the clique-width of the incidence graph is tractable, then model counting is tractable. A detailed comparison between expression treewidth and clique-width is left as future work.

7. REFERENCES

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8:277–284, April 1987.

[2] L. Beineke and R. Pippert. The number of labeled k-dimensional trees. *Journal of Combinatorial Theory*, 6(2):200 – 205, 1969.

[3] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *STOC*, pages 226–234, 1993.

[4] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.

[5] R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

[6] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[7] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[8] N. N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. In *PODS*, pages 203–214, 2010.

[9] A. Darwiche. *Modeling and Reasoning with Bayesian Network*. Cambridge University Press, April 2009.

[10] A. Ferrara, G. Pan, and M. Y. Vardi. Treewidth in verification: Local vs. global. In *LPAR*, pages 489–503, 2005.
[11] E. Fischer, J. A. Makowsky, and E. V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.
[12] M. C. Golumbic, A. Mintz, and U. Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Applied Mathematics*, 154(10):1465–1477, 2006.
[13] G. Gottlob, R. Pichler, and F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.
[14] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.
[15] V. Gurvich. Repetition-free boolean functions. *Uspekhi Mat. Nauk*, 32:183–184, 1977.
[16] J. Huang and A. Darwiche. Using dpll for efficient obdd construction. In H. H. Hoos and D. G. Mitchell, editors, *Theory and Applications of Satisfiability Testing*, volume 3542 of *Lecture Notes in Computer Science*, pages 157–172. Springer Berlin / Heidelberg, 2005.
[17] A. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, pages 323–334, 2010.
[18] A. Jha and D. Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. In *ICDT*, pages 162–173, 2011.
[19] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302 – 332, 2000.
[20] D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.
[21] D. Olteanu and J. Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, pages 389–402, 2009.
[22] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, September 1988.
[23] S. Roy, V. Perduca, and V. Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.
[24] P. Scheffler. *Die Baumweite von Graphen als ein Ma8 Rir die Kompliziertheit algorithmischer Probleme*. PhD thesis, Akademie der Wissenschafien der DDR, Berlin, 1989.
[25] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. In *VLDB*, 2010.
[26] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
[27] L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.
[28] I. Wegener. BDDs–design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.

APPENDIX

We first show another way to characterize treewidth through *partial k-trees* [2]. We will present a criterion based on that but motivated more by the *variable elimination* [22] algorithm used in probabilistic inference. Let π be a permutation over $V(G)$. Define the following process that starts with G, pops the first vertex from π and removes it from G, while connecting all its neighbors to form a clique. Continue till π and G are empty. At each stage, when a variable v is eliminated we generate a clique of size the number of neighbors of v. We say π achieves a width of k, if the size of maximal clique generated by the above process is k. Then

PROPOSITION .1. *$twd(G)$ is the minimum width achievable over all possible permutations of $V(G)$.*

The above is well known, but we still give a proof below for the sake of completeness.

PROOF. Given a permutation π achieving a width k, construct a tree-decomposition inductively as follows : Take the vertex $\pi(1)$ and make a node X_1 consisting of $\pi(1)$ and all its neighbors. Now let $(\bar{Y}, T)$ be the tree-decomposition of the graph obtained after eliminating $\pi(1)$. We claim it has a node Y_j s.t. all the neighbors of X_1 are present in Y_j. This is actually a general phenomenon : for any clique in a graph, there exists a node in its tree-decomposition that contains all the vertices in the clique. It can be proven inductively : for cliques of size 2 its true by definition. For a clique of size $k+1$, we know that there are nodes that contain cliques of size k in it, and they must be connected, giving rise to a cycle, a contradiction. Hence, to get the tree-decomposition of G, we just connect X_1 to Y_j.

On the other hand, suppose we are given a tree decomposition $(\bar{X}, T)$ of G, we define the variable order π as follows : start with any leaf node X_j in T and add at the end of π all the vertices in X_j that are not present in any other node of T and then remove X_j from T. Keep repeating until T is empty. Note that whenever we remove a node v from X_j, its set of neighbors is at most $|X_j| - 1$. As far as making a clique out of neighbors of v goes, that happens implicitly in tree-decompositions, since they are all present in X_j. □

PROOF OF PROP. 2.5. We first prove that any expression according to Eq.(2) can be expressed as an expression according to Eq.(3) with the same treewidth.
Let G be the expression according to Eq.(2). Note that every $\wedge, \vee$ operator over several variables can be expressed in terms of binary operators, because of the associative nature of these operators. So, we express any $\vee$ node, say v (with greater than 2 children) : $X_{i1} \vee X_{i2} \vee \ldots X_{il}$ as $X_{i1} \vee (X_{i2} \vee \ldots) \vee X_{il}) \cdots)$. So v has been split into $l-1$ nodes $\bar{w}$: $w_j = X_{ij} \vee w_{j+1}, j < l-1$ and $w_{l-1} = X_{i(l-1)} \vee X_{il}$. Similarly for $\wedge$. Call the new expression G'. It clearly represents the same formula F. We now show it has the same treewidth as G.

In the above construction each internal node v may be split into multiple internal nodes $R(v) = \bar{w}$. For leaves and nodes with at most two children, $R(v)$ only contains v. Now given any variable elimination order π over V achieving treewidth t, we construct an order π' over $\bigcup_{v \in V} R(v)$, s.t. π' achieves treewidth t for G'. We iterate through nodes v according to order π : we copy v into π' if v has at most two children or is a leaf; else we add to π' : $w_{l-1}, w_{l-2}, \ldots, w_1$ in that order.

Now consider the graphs starting from G, G', generated by following the elimination order π, π'. We know that the size of maximum clique generated by π over G is t. We need to show the same for π' over G'. Assume it holds until v is eliminated. Let $G_{\geq v}, G'_{\geq v}$ be the graphs just before elimination of v. Now we eliminate $v_{l-1}, v_{l-2}, \ldots, v_1$ in that order for G' and we need to show the size of maximum clique generated is $|N(v)|$, where $N(v)$ is the neighbors of v in $G_{\geq v}$. First observe that the union of neighbors of $v_j, j \leq l-1$ is the same as $N(v)$. This is true by definition in the beginning when no nodes have been eliminated and whenever we

eliminate a neighbor of v, which is say a neighbor of v_j, and connect its neighbors to v, in G' they are connected to v_j. Hence now by inspection as we eliminate $\bar{v}$ at the end, the size of clique is $|N(v)|$, while at any stage in between its less than that. Hence proved.

Now assume G be an expression according to Eq.(3). Clearly if the gates are all $\wedge, \vee, \neg$, then we are done, since the expression also follows Eq.(2). Otherwise, for any $a \otimes b$, we express the $\otimes \in \mathbb{B}_2$ in terms of these gates. For e.g., $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$. We introduce a constant number of new interior gates by this expansion, and these interior gates aren't connected to any other nodes of G, except for a, b. Hence the treewidth only increases some constant times. □

PROOF OF TH. 3.9 CONTD... Here, we prove that if $Q \notin IF^{\neq}$, then it has no *OBDD* of polynomial size. To do this we need to extend the hardness proof from [18] that showed that if $Q \notin IF$ then it has no *OBDD* of polynomial size. Before we review that proof, we need an alternate definition of *inversion* :

Let $Q = q_1 \vee \ldots \vee q_k$ be a query in DNF. The *unification graph* G has as nodes all pairs of variables (x, y) that co-occur in some atom, and has an edge between (x, y) and (x', y') if : suppose x, y co-occur in g , x', y' co-occur in g', then g and g' are over the same relation symbol and x, y appear at the same positions in g as x', y' in g'. (In other words, g and g' are unifiable, and the unification equates $x = x'$ and $y = y'$.). Given $x, y \in Vars(q_i)$, denote $x \succ y$ if $at(x) \not\subseteq at(y)$.

DEFINITION .2 (INVERSION). *An inversion in Q is a path of length ≥ 0 in G from a node (x, y) to a node (x', y') s.t. $x \succ y$ and $x' \prec y'$. If no such path exists, we say Q is inversion-free.*

We briefly review the proof from [18] next :
For $k \geq 1$, define the following queries :

$$\begin{aligned} h_{k0} &= R(x_0), S_1(x_0, y_0) \\ h_{ki} &= S_i(x_i, y_i), S_{i+1}(x_i, y_i) \qquad i = 1, k-1 \\ h_{kk} &= S_k(x_k, y_k), T(y_k) \end{aligned}$$

The *complete bipartite graph* of size n is the following database D over the vocabulary of h_k: relation R has n tuples : $R(a_1), R(a_2), \ldots, R(a_n)$, relation T has n tuples $T(b_1), T(b_2)$, $\ldots, T(b_n)$ and each relation S_i has n^2 tuples $S_i(a_j, b_l)$, for $i = 1, k$, and $j, l = 1, n$. We can show that if the same variable order Π is used to compute all queries h_{k0}, h_{k1}, $\ldots$, h_{kk} over a complete bipartite data, then at least one of these $k+1$ *OBDD* has exponential size.

PROPOSITION .3. *Let D be the complete bipartite graph of size n, and fix any ordering Π on the corresponding Boolean variables. For any $i = 0, k$, let n_i be the size of some Π-OBDD for the lineage of h_{ki} on D. Then $\sum_{i=0}^{k} n_i > k \cdot 2^{\frac{n}{2k}}$.*

Then given any query with inversion of length k(i.e., not inversion-free) over database D : by suitably setting some variables to 0/1 in the lineage of $F(Q, D)$, we get the lineage of $h_{ki}, i = 0 \ldots k$ over a complete bipartite graph of size $O(D|)$. This implies that if there were a variable order Π s.t., $OBDD_{\Pi}(F(Q, D))$ was polynomial, then we could get a polynomial size OBDD for each of $h_{ki}, i = 0 \ldots k$ over a complete bipartite data with the same variable order, a contradiction. It is this last part that we need to show holds for $IF^{\neq}$ too.

Write $Q = \bigvee(q_j \wedge p_j$ in DNF, and let $(x_0, y_0), (x_1, y_1)$, $\ldots, (x_k, y_k)$ be an inversion in Q. Assume w.l.o.g. that the inversion is of minimal length: this implies there exist atoms $r, s_1, s'_1, \ldots, s_k, s'_k, t$ with the following properties: $r \in at(x_0) - at(y_0)$, $t \in at(y_k) - at(x_k)$, and for every $i = 1, k$, s_i contains x_{i-1}, y_{i-1}, s'_i contains x_i, y_i, they unify, and the unification equates $x_{i-1} = x_i$ and $y_{i-1} = y_i$. In particular, the atoms s_i and s'_i have the same relation symbol. Assume that x_i, y_i are variables in the query q_{j_i}, for $i = 0, k$. We assume that these k queries are distinct: if not, simply create a fresh copy of the query, creating new copies of its variables. Thus, q_{j_0} contains the atoms r, s_1, query q_{j_1} contains the atoms s'_1, s_2 and so on. Next, we perform variable substitutions in the queries $q_{j_0}, \ldots, q_{j_k}$ in order to equate all variables in s_i and s'_i, except for $x_{i-1}, y_{i-1}, x_i, y_i$. In other words, all atoms along the inversion path have the same variables, except for the variables forming the actual inversion. For example, if the queries were $R(x_0, u_0), S_1(x_0, y_0, u_0)$; $S_1(x_1, y_1, u_1), S_2(x_1, y_1, u_1, v_1)$; $S_2(x_2, y_2, u_2, v_2), \ldots$ then we equate $u_0 = u_1 = u_2 = \ldots$ and $v_1 = v_2 = \ldots$ This is possible in general because Q is ranked: we only equate variables between q_{j_i} and q_{j_l}, but not within the same q_{j_i}. We now construct the database D as follows. Its active domain consists of all constants $a_1, \ldots, a_n, b_1, \ldots, b_n$ and all variables $z \in Vars(q_{j_i})$ s.t. $z \neq x_i$, $z \neq y_i$, for $i = 0, k$. For each $i = 0, k$, and each $j = 1, n$, $l = 1, n$, let $q_{j_i}[a_j, b_l]$ denote the set of tuples obtained by substituting x_i with a_j and y_i with b_l. Define D to be the union of all these sets: $D = \bigcup_{i,j,l} q_{j_i}[a_j, b_l]$. Because of our earlier variable substitutions, $s_i[a_j, b_l]$ and $s'_i[a_j, b_l]$ are the same tuple: this tuple corresponds to the tuple $S_i(a_j, b_l)$ in the bipartite graph. Similarly, $r(a_j)$ and $t(b_l)$ correspond to the tuples $R(a_j)$ and $T(b_l)$ in the bipartite graph. Thus, the bipartite graph D_0 is isomorphic to a subset of the database D. Consider now any OBDD for $F(Q, D)$, over a fixed variable ordering Π. We can obtain an OBDD for h_{ki} for every $i = 0, k$ as follows. Assume $0 < i < k$. Then we keep unchanged the Boolean variables corresponding to $S_i(a_j, b_l)$ and $S_{i+1}(a_j, b_l)$, (that is, the atoms $s'_i[a_j, b_l]$ and $s_{i+1}[a_j, b_l]$). All other Boolean variables corresponding to tuples in $q_{j_i}[a_j, b_j]$ are set to `true`; all remaining Boolean variables are set to `false`. Then the lineage $F(Q, D)$ becomes the lineage $F(D_0, h_{ki})$. To see why inequalities don't matter, note that an inequality $x_i \neq y_i$, is always true as their domains are different. Any other inequality of form $x_i \neq z$ is true since domain of x_i is more than 1, which is true in our case. The same is true for inequalities of the form $z_1 \neq z_2$. In other words all the inequalities are `true` and the lineage remains unchanged. The case $i = 0$ is similar (here we keep unchanged the Boolean variables corresponding to $R(a_j)$ and $S_1(a_j, b_l)$), and so is the case $i = k$. Thus, we obtain $k+1$ OBDD's for all queries h_{ki}, and all use the same variable order Π. □