Using OBDDs for Efficient Query Evaluation on Probabilistic Databases

Dan Olteanu and Jiewen Huang

Oxford University Computing Laboratory, UK

Abstract. We consider the problem of query evaluation for tuple independent probabilistic databases and Boolean conjunctive queries with inequalities but without self-joins. We approach this problem as a construction problem for ordered binary decision diagrams (OBDDs): Given a query q and a probabilistic database D, we construct in polynomial time an OBDD such that the probability of $q(D)$ can be computed linearly in the size of that OBDD. This approach is applicable to a large class of queries, including the *hierarchical* queries, i.e., the Boolean conjunctive queries without self-joins that admit PTIME evaluation on any tuple-independent probabilistic database, hierarchical queries extended with inequalities, and non-hierarchical queries on restricted databases.

1 Introduction

Recently there has been renewed interest in probabilistic databases [2, 10, 20, 5, 6, 1] due to important applications that systems for representing uncertain information have, such as data cleaning, data integration, and scientific databases.

In this paper we study the following evaluation problem: given a Boolean conjunctive query q without self-joins and with inequalities and a tuple-independent probabilistic databases D, compute the probability of $q(D)$.

Dalvi and Suciu's seminal work [5] on the evaluation of conjunctive queries without self-joins on tuple-independent probabilistic databases shows that the complexity of query evaluation is either PTIME or #P-hard. In case of PTIME queries, also called hierarchical [6], there exists an evaluation method that rewrites them into linear-size SQL queries (called safe plans) that compute the probability of the distinct answer tuples. Such SQL rewritings use aggregates to eagerly eliminate duplicates and compute the probability of distinct tuples in projections of the input and temporary tables. The addition of aggregates severely restricts the search space for good query plans to compute the answer tuples: In most cases it enforces unoptimal join orderings and each of these aggregates requires sorting. We also note that this rewriting approach cannot be naturally extended to cope with queries beyond the hierarchical ones.

In this paper we devise a new method for the aforementioned evaluation problem. Our method is rooted in the following two observations that relate query evaluation on probabilistic databases, #SAT procedures, and knowledge compilation. First, the probability of a query q on a probabilistic database D is the probability of the Boolean expression $\phi_{q,D}$ *associated* with q and D; such Boolean expressions encode the (provenance) information on which input tuples contribute to which answer tuples. Second, the probability of $\phi_{q,D}$ can be computed by compiling it into a propositional theory with PTIME model counting (and thus probability computation). Our approach is to compile $\phi_{q,D}$ into *reduced ordered binary decision diagrams* (OBDDs). Boolean expressions are commonly represented using OBDDs, as it is the case in hardware verification and model checking [4], program analysis [15], and probabilistic logic programming [18].

We show that OBDDs are effective in handling Boolean expressions of interest. In contrast to the approach of Dalvi and Suciu, our approach is more general as it covers the *orthogonal* tractable classes of both hierarchical queries and Boolean expressions of bounded treewidth, which are associated with probabilistic databases and non-hierarchical queries, and a large tractable class of conjunctive queries extended with inequalities.

The key technical challenge of our method is to efficiently find *good variable orders*, under which Boolean expressions associated with queries and probabilistic databases can be compiled into OBDDs in polynomial time.

The contributions of this article are as follows:

- We revisit the problem of query evaluation for conjunctive queries on tuple-independent probabilistic databases and connect it to the OBDD construction problem.
- We show that the expression $\phi_{q,D}$ associated with any hierarchical query q and tuple-independent probabilistic database D, can be brought into a special factored form, where each of its variables occurs exactly once. It then follows that such expressions can be efficiently compiled into OBDDs, whose sizes are linear in the number of their variables. This guarantees the robustness of our method.
- We define a large tractable class of queries with inequalities. Queries in this class can be represented as trees where nodes are hierarchical queries and each edge that connects two nodes for queries A and B represents one inequality on variables occuring in all subgoals of A and B, respectively.
- Within the #P-hard class of conjunctive queries, we identify one subclass that remains in PTIME under certain assumptions about the database. By relating the complexity of query evaluation to that of OBDD construction for arbitrary Boolean expressions, we are able to carry over results that bound the exponent of the evaluation time to the treewidth of such expressions.

To the best of our knowledge, this paper is the first to develop a robust framework based on OBDDs to efficiently evaluate queries on probabilistic databases. Similar in spirit to the approach of this paper, previous work [14] of the first author employs knowledge compilation techniques for probability computation of conjunctive queries on *arbitrary* probabilistic databases, but *without* polynomial-time guarantees. Follow-up work [17] of the same authors applies the results of this paper to implement in PostgreSQL a low-level query plan operator for probability computation, and shows experimentally that our method can outperform the method of Suciu and Dalvi by orders of magnitude.

R	A	B
x_1	a_1	b_1
x_2	a_2	b_1
x_3	a_2	b_2
x_4	a_3	b_3

S	A	C
y_1	a_1	c_1
y_2	a_1	c_2
y_3	a_2	c_1
y_4	a_4	c_2

T	D
z_1	c_1
z_2	c_2
z_3	c_3

Fig. 1. A tuple-independent probabilistic database over $\{R(A, B), S(A, C), T(D)\}$.

2 Preliminaries

We next recall the notions of probabilistic databases, conjunctive queries, and ordered binary decision diagrams.

2.1 Tuple-independent Probabilistic Databases

Let a finite set $\mathbf{X}$ of (independent) random Boolean variables and a probability distribution over their assignments given by a function P, i.e., $\forall x \in \mathbf{X} : P(x) + P(\overline{x}) = 1$. A *probabilistic relation* R over a schema and variable set $\mathbf{X}$ is a set of tuples over that schema, such that each tuple is associated with a distinct variable from $\mathbf{X}$. We denote by $Vars(R) \subseteq \mathbf{X}$ the set of variables of R. A *probabilistic database*, or database for short, is a set of probabilistic relations. Fig.1 gives such a database, where for instance $Vars(R) = \{x_1, x_2, x_3, x_4\}$.

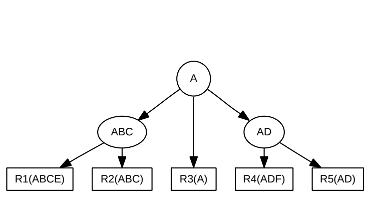
The set of *possible worlds* is defined by the finite set of truth assignments of all variables from $\mathbf{X}$. There is a one-to-one correspondence between possible worlds and database instances. To obtain one instance, we fix a truth assignment f, and then process each relation R_i tuple by tuple. A tuple t with variable $\phi(t)$ is in R_i if $f(\phi(t))$ is true. For instance, the truth assignment that maps x_1, y_1, and z_1 to true and all remaining variables to false, defines the database instance where $R = \{(a_1, b_1)\}$, $S = \{(a_1, c_1)\}$, and $T = \{(c_1)\}$. The probability of this world is the product of the probabilities of x_1, y_1, and z_1 being true, and of the probabilities of the remaining variables being false.

2.2 Conjunctive Queries with Inequalities and without Self-Joins

We consider Boolean conjunctive queries with negated equalities but without self-joins. We write queries using the Datalog notation: q :- $g_1, \ldots, g_n$ defines a query q where its body is a conjunction of n distinct positive relational predicates, called *subgoals*. A subgoal has the form $R(A_1, \ldots, A_k)$, where R is a relation name and A_1 to A_k are query variables. By $sg(A_i)$ we denote the set of subgoals of query variable A_i. An *eq-join variable* occurs in more than one subgoal. Inequality joins are expressed using inequality conditions, e.g., $B \neq C$ with query variables B and C occurring in some subgoals.

We partition the conjunctive queries into *hierarchical* and non-hierarchical [7]: The hierarchical queries admit polynomial-time evaluation, whereas the non-hierarchical ones are #P-hard in general [5].

Definition 1 ([7]). *A conjunctive query is hierarchical if for any two variables, either their sets of subgoals are disjoint, or one set is contained in the other.*



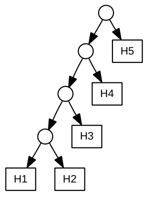


Fig. 2. (left) Hierarchical query of Ex.1 and (right) IHQ$^{\neq}$ query of Ex.2.

Each connected component of a hierarchical query has at least one query variable that occurs in all subgoals. Following [7], we call such variables *maximal*. We represent hierarchical queries as trees, where the inner nodes are the join variables of the children and the leaves are query subgoals. The root is then the set of maximal variables in case of connected queries, or the empty set otherwise. Each inner node stands for a relation, which corresponds to the subquery of the tree rooted at that node, and can be realized as the natural join of the node's children followed by a projection on the node's join variables.

Example 1. The following query is hierarchical and the variable A is maximal:

$$h\,{:}\text{-}\,R_1(A,B,C,E), R_2(A,B,C), R_3(A), R_4(A,D,F), R_5(A,D).$$

Fig.2 gives its tree representation. If we remove A from either R_1 or R_2, we obtain a non-hierarchical query, because $sg(A) - sg(B) \neq \emptyset \neq sg(B) - sg(A)$.$\Box$

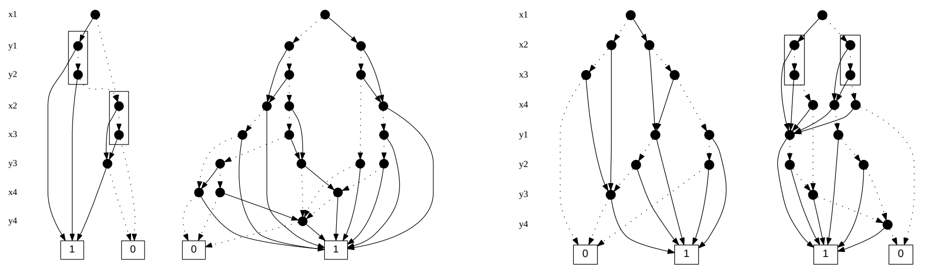
We also consider a class of conjunctive queries with inequalities, which we show in Section 4 to be tractable.

Definition 2. *An IHQ$^{\neq}$ query is either hierarchical, or a join of two independent IHQ$^{\neq}$ queries using an inequality predicate on maximal query variables. Two queries are independent if they use disjoint sets of relations.*

IHQ$^{\neq}$ queries have no cycles containing inequalities. We use here a tree representation that cannot distinguish unconnected hierarchical queries from IHQ$^{\neq}$ queries. Consider a partial order on the hierarchical subqueries of a IHQ$^{\neq}$ query q such that if one subquery is joined with n others, then it occurs after all subqueries joined with at most $n-1$ others (the acyclicity ensures the existence of such orders). We construct a binary left-deep tree representation of q by adding in the ordered subqueries from right to left. The leaves of such a tree represent the hierarchical subqueries and the inner nodes are labeled with empty sets. The leaves are then replaced by the tree representations of the subqueries.

Example 2. The IHQ$^{\neq}$ query

$$\begin{aligned}
q\,{:}\text{-}\,&R_1(A,B), R_2(A,C), && (H_1)\\
&U(H,I), && (H_2)\\
&T(F,G), && (H_3)\\
&S_1(C,D,E), S_2(C,D), D \neq A, C \neq F, && (H_4)\\
&V(J,K), J \neq I, K \neq C. && (H_5)
\end{aligned}$$



(a) Good variable order π_1 for eq-joins (b) Good variable order π_2 for neq-joins
OBDDs: (a) left (ϕ_{eq}, π_1), (a) right (ϕ_{neq}, π_1), (b) left (ϕ_{eq}, π_2), (b) right (ϕ_{neq}, π_2).
The expressions ϕ_{eq} and ϕ_{neq} are given in Example 3.

Fig. 3. Eq-joins and neq-joins have different good variable orders.

consists of five hierarhical queries (denoted by H_1 to H_5 above). Fig.2 gives the tree representation of q corresponding to the order H_1, H_2, H_3, H_4, H_5. For space reasons, we do not replace the leaves H_i by their tree representations. $\square$

The query evaluation follows the standard semantics with the addition that each tuple t is associated with a Boolean expression over random variables [13], as shown below for product, selection, and projection:

$$\begin{aligned} Q_1 \times Q_2 &= \{(t_1 \circ t_2, \phi_1\phi_2) \mid (t_1, \phi_1) \in Q_1, (t_2, \phi_2) \in Q_2\} \\ \sigma_{cond}(Q) &= \{(t, \phi) \mid (t, \phi) \in Q, cond(t)\} \\ \pi_{\bar{A}}(Q) &= \{(t.\bar{A}, \phi) \mid (t, \phi) \in Q\} \end{aligned}$$

The expression associated with q and D, denoted by $\phi_{q,D}$, is the disjunction of the monotone expressions of the tuples in $q(D)$: $\phi_{q,D} := \sum_{(t_i,\phi_i)\in q(D)}(\phi_i)$. The *size* of an expression $\phi_{q,D}$, denoted by $|\phi_{q,D}|$, is the product of the number of its clauses (equal to the number of tuples in the answer $q(D)$) and the number of variables per clause (equal to the number the subgoals of q).

Proposition 1 ([5]). *For any query q and probabilistic database D, it holds that $P(q(D)) = P(\phi_{q,D})$.*

Example 3. Consider the Boolean queries

$$q_{eq} :\!- R(A, B), S(A, C) \qquad\qquad q_{neq} :\!- R(A, B), S(C, D), A \neq C$$

The expressions ϕ_{eq} and ϕ_{neq} are (in an easier to follow factored form)

$$\begin{aligned} \phi_{eq} &= x_1(y_1 + y_2) + (x_2 + x_3)y_3 \\ \phi_{neq} &= x_1(y_3 + y_4) + (x_2 + x_3)(y_1 + y_2 + y_4) + x_4(y_1 + y_2 + y_3 + y_4) \end{aligned}$$

2.3 Ordered Binary Decision Diagrams

Reduced ordered binary decision diagrams (OBDDs) are commonly used to represent compactly large Boolean expressions [16].

The idea behind OBDDs is to decompose Boolean expressions using variable elimination and to avoid redundancy in the representation. The decomposition step is normally based on exhaustive application of Shannon's expansion: Given a Boolean expression ϕ and one of its variables x, we have $\phi = x \cdot \phi\mid_x + \bar{x} \cdot \phi\mid_{\bar{x}}$, where $\phi\mid_x$ and $\phi\mid_{\bar{x}}$ are ϕ with x set to true and false, respectively. The order of variable eliminations is a total order π on the set of variables of ϕ, called *variable order*. An OBDD for ϕ is uniquely identified by the pair (ϕ, π).

OBDDs are represented as directed acyclic graphs (DAG), with two terminal nodes representing the constants 0 (false) and 1 (true), and non-terminal nodes representing variables. Each node for a variable x has two outgoing edges corresponding to the two possible variable assignments: a high (solid) edge for $x = 1$ and a low (dashed) edge for $x = 0$. To evaluate the expression for a given set of variable assignments, we take the path from the root node to one of the terminal nodes, following the high edge of a node if the corresponding input variable is true, and the low edge otherwise. The terminal node gives the value of the expression. The non-redundancy is what makes OBDDs usually more compact than the textual representation of Boolean expressions: a node n is redundant if both its outgoing edges point to the same node, or if there is a node for the same decision variable and with the same children as n.

The choice of variable order can greatly influence the size of the OBDD.

Definition 3. *A variable order π is good for an expression ϕ if it can be computed from ϕ in PTIME and the OBBD (ϕ, π) has size polynomial in $|\phi|$.*

Some expressions do not admit good orders, either because they do not admit polynomial-size OBDDs, or because computing orders for such OBDDs is NP-hard [16]. In this paper, we nevertheless show that expressions associated with hierarchical queries and even with IHQ$^{\neq}$ queries admit good variable orders. Additionally, although not obvious in general, we are able to construct polynomial-size OBDDs in an *output-sensitive* manner and hence in PTIME.

Example 4. Fig. 3 depicts OBDDs for the expressions ϕ_{eq} and ϕ_{neq} of Example 3 under two distinct variable orders π_1 and π_2. The variable order π_1 is good for ϕ_{eq} as the size of the OBDD (ϕ_{eq}, π_1) is linear in the number of ϕ_{eq}'s variables. We show later that the variable order π_2 is good for ϕ_{neq}. □

OBDDs can be maniputated efficiently. We exemplify here with linear-time probability computation, given a probability distribution over the OBDD variables. Fig. 4 gives the procedure **prob** to this effect. We consider that for each OBDD node n, its variable is accessible by `n.v`, its probability by `n.p`, and its children by `n.high` and `n.low`. The probability value is initialized to 0 for terminal node 0, to 1 for terminal node 1, and to -1 for the remaining nodes. The probability of the OBDD is the probability of its root node, and the probability of any inner node `n` is the sum of the probabilities of their children weighted by the probabilities of the corresponding assignments of the decision variable `n.v`. Because we do a constant number of operations per node, we have that

```
prob (Node n)
  if (n.p = – 1) then n.p := P(n.v) * prob(n.high) + (1 – P(n.v)) * prob(n.low);
   return n.p;
end
```

Fig. 4. Computing the probability of an OBDD.

Proposition 2. *The probability of the OBDD* (ϕ, π) *for an expression* ϕ *and a variable order* π *can be computed in time* $O(|(\phi, \pi)|)$*.*

From Query Evaluation to OBDD Construction. The problem of query evaluation on (not necessarily tuple-independent) probabilistic databases and the OBDD construction problem are closely connected. In particular, an efficient solution to the latter guarantees an efficient solution to the former. The connection follows in two steps: A reduction from the evaluation problem to probability computation of expressions over random Boolean variables, followed by a reduction from the latter problem to the problem of construction and probability computation of OBDDs.

Proposition 3. *For any query* q*, database* D*, and variable order* π *of* $\phi_{q,D}$*, it holds that* $P(q(D)) = P((\phi_{q,D}, \pi))$*.*

By Proposition 2, we can linearly reduce the query evaluation problem to the problem of OBDD construction.

Corollary 1. *Let query* q *and probabilistic database* D*. If there is a good variable order* π *such that the OBDD* $(\phi_{q,D}, \pi)$ *can be constructed in time polynomial in* $|\phi_{q,D}|$*, then* $P(q(D))$ *can be computed in PTIME.*

3 Hierarchical Queries

The hierarchical queries are the Boolean conjunctive queries without self-joins that admit PTIME evaluation on any tuple-independent probabilistic database [6]. The main result of this section is that

Theorem 1. *For any hierarchical query* q *and database* D *there is a good variable order* π *for* $\phi_{q,D}$*. In particular,* π *can be computed in time* $O(|\phi_{q,D}| \log^2 |\phi_{q,D}|)$ *and, given* π*, the OBDD* $(\phi_{q,D}, \pi)$ *can be computed in time* $O(|Vars(\phi_{q,D})|)$*.*

Example 5. Consider the eq-join query q_{eq} :- $R(A, B), S(A, C)$ of Example 3 and the database of Fig. 1, where R and S are partitioned according to the A-values. By the semantics of eq-join, the tuples of the a_1-partitions of R and S are paired independently of the a_2-partitions. We thus generate a disjunction of conjunctions representing *all* pairs of variables from the two partitions, i.e., x_1 is paired with y_1 and y_2, and x_2 and x_3 are paired with y_3. This information is made explicit in the factored form of ϕ_{eq} given in Example 3.

A good variable order makes use of the independence of conjunctions across partitions. We partition the set of variables of ϕ_{eq} in independent sets $\{x_1, y_1, y_2\}$

$$\begin{array}{ll}
\textbf{type}\ (\text{query } q) & = \tau(q, \emptyset) \\
\tau\ (\text{inner node } \bar{A}(X_1, \ldots, X_n),\ L) & = \textbf{ite}(L = \bar{A},\ \tau(X_1, \bar{A}) \circ \ldots \circ \tau(X_n, \bar{A})) \\
\tau\ (\text{leaf node } R(\bar{A}),\ L) & = \textbf{ite}(L = \bar{A}, Vars(R)) \\
\textbf{ite}\ (\text{Cond, t}) & = \textbf{if}\ \text{Cond}\ \textbf{then}\ t\ \textbf{else}\ (t)^*
\end{array}$$

Fig. 5. Deriving VO-types from queries represented in tree form.

and $\{x_2, x_3, y_3\}$, and choose a total order on the sets. We also exploit the fact that, within any of these sets, all variables from the partition of R are combined with all variables from the corresponding partition of S. The good orders for ϕ_{eq} thus correspond to any permutation of elements within each (nesting or nested) set in $\{\{x_1, \{y_1, y_2\}\}, \{\{x_2, x_3\}, y_3\}\}$. Fig. 3 gives one of these good orders: $\pi_1 = x_1 y_1 y_2 x_2 x_3 y_3$, which induces an OBDD for ϕ_{eq} whose size is linear in the number of variables. The boxes surrounding the subgraphs for the expressions $y_1 + y_2$ and $x_2 + x_3$ highlight that under such good orders each of them can be treated as one variable (each box has one parent and two distinct children). $\square$

We next generalize our reasoning from Example 5. For a given hierarchical query q, we first derive a class of variable orders, or VO-type for short, that captures good variable orders, and then use the VO-type and the expression $\phi_{q,D}$ associated with q and D to create a good variable order.

Definition 4. *A VO-type is defined inductively as follows:*

- *A set* $\mathbf{X}$ *of variables is a VO-type that defines all variable orders consisting of one variable of* $\mathbf{X}$*;*
- *A reflexive transitive closure* α^* *of a VO-type* α *is a VO-type that defines all variable orders obtained by concatenating zero or more variable orders of* α*;*
- *A (unordered) concatenation* $\alpha\beta$ *of two VO-types* α *and* β *is a VO-type that defines all variable orders obtained by concatenating a variable order of* α *(*β*) and a variable order of* β *(resp.* α*).*

A set $\mathbf{X}$ *can only occur once in a VO-type.*

Example 6. Let $\mathbf{X} = \{x_1, x_2, x_3, x_4\}$ and $\mathbf{Y} = \{y_1, y_2, y_3, y_4\}$. The VO-types of the variable orders of Fig. 3 are $(\mathbf{X}^*\mathbf{Y}^*)^*$ and $\mathbf{X}^*\mathbf{Y}^*$, respectively. $\square$

Fig. 5 gives the function **type** that constructs a VO-type from the tree representation of a query q. While traversing the tree top-down, we keep the query variables of the parent node (which includes the variables of all the ancestors) in L; initially, $L = \emptyset$. For a query subgoal $R(\bar{A})$, we create a VO-type $Vars(R)$ or $(Vars(R))^*$. The former case occurs when $\bar{A}$ represents the parent variables L, and thus there is one tuple (and thus one variable) per distinct $\bar{A}$-value. Otherwise, there may be several tuples (variables) per distinct $\bar{A}$-value (due to the projection at the parent node on the proper subset L of $\bar{A}$), and hence the exponent (*). In case of an inner node, we recursively compute the VO-types for the children and then concatenate them in *any* order. Distinction on $\bar{A} = L$ applies here as well. Note that $\bar{A} = \emptyset$ covers the case of hierarchical subqueries that are unconnected or connected using inequalities (IHQ$^{\neq}$ queries discussed later). In both cases, we treat such subqueries independently.

$$\begin{array}{|lll|}\hline \mathbf{vo}\ (\alpha^*, \phi) & = & \mathbf{let}\ \phi_1, \ldots, \phi_n \text{ be a maximal independent partitioning of } \phi\ \mathbf{in} \\ & & \quad \mathbf{vo}\ (\alpha, \phi_1) \circ \ldots \circ \mathbf{vo}\ (\alpha, \phi_n) \\ \mathbf{vo}\ (\alpha\beta, \phi) & = & \mathbf{vo}\ (\alpha, \phi \text{ restricted to } Vars(\alpha)) \circ \mathbf{vo}\ (\beta, \phi \text{ restricted to } Vars(\beta)) \\ \mathbf{vo}\ (\mathbf{X}, \phi) & = & \text{the only variable in } \phi \\ \hline \end{array}$$

Fig. 6. Deriving good variable orders for Boolean expressions wrt given VO-types.

Example 7. The query of Example 1 has the VO-type $((\mathbf{X}_1^*\mathbf{X}_2)^*\mathbf{X}_3(\mathbf{X}_4^*\mathbf{X}_5)^*)^*$, where $\mathbf{X}_i = Vars(R_i)$. The IHQ$^{\neq}$ query of Example 2 has the VO-type $(Vars(R_1)^*Vars(R_2)^*)^*Vars(U)^*Vars(T)^*\ (Vars(S_1)^*Vars(S_2))^*Vars(V)^*$.□

The VO-type of a hierarchical query q is also useful for bringing the expression $\phi_{q,D}$ in a factored form where each variable of $\phi_{q,D}$ occurs exactly once.

Definition 5. *A DNF expression ϕ can be factored according to a VO-type*

- $\mathbf{X}$ *if ϕ is in one variable and that variable occurs in the set $\mathbf{X}$;*
- α^* *if there exist DNF expressions $\phi_1, \ldots, \phi_n$ that can be factored according to α, $\phi = \phi_1 + \ldots + \phi_n$ and $\forall 1 \leq i < j \leq n : Vars(\phi_i) \cap Vars(\phi_j) = \emptyset$;*
- $\alpha\beta$ *if there exist DNF expressions ϕ_1 and ϕ_2 that can be factored according to α and β, respectively, $\phi = (\phi_1)(\phi_2)$, and $Vars(\phi_1) \cap Vars(\phi_2) = \emptyset$.*

Example 8. As shown in Example 3, the expression ϕ_{eq} can be factored according to the VO-type $(Vars(R)^*Vars(S)^*)^*$ of q_{eq}: Some variables of $Vars(R)$ are paired with some variables of $Vars(S)$, and the same may apply to further independent sets of variables in $Vars(R)$ and $Vars(S)$. □

Lemma 1. *For any hierarchical query q and database D, $\phi_{q,D}$ can be factored according to VO-type **type**(q).*

Fig. 6 gives the function **vo** that computes good variable orders. This function uses pattern matching on the structure of VO-types.

In case of VO-types α^*, the variable order is a concatenation of variable orders for α. Because these variable orders use disjoint sets of variables, we compute the *maximally independent partitioning* of the input expression ϕ and continue on each partition independently. A partitioning $\phi_1, \ldots, \phi_n$ of $\phi_{q,D}$ is independent if $\phi_{q,D} = \phi_1 + \ldots + \phi_n$ and $\forall 1 \leq i \neq j \leq n : Vars(\phi_i) \cap Vars(\phi_j) = \emptyset$. An independent partitioning of $\phi_{q,D}$ is maximal if $\phi_{q,D}$ has no finer partitioning. For instance, consider again the expressions ϕ_{eq} and ϕ_{neq} of Example 3. A maximal independent partitioning of ϕ_{eq} is given by $x_1y_1 + x_1y_2$ and $x_2y_3 + x_3y_3$. The expression ϕ_{neq} has no maximal independent partitioning but itself.

In case of concatenated VO-types $\alpha\beta$, we recursively compute the variable order for α independently of β on the restrictions of ϕ computed by eliminating all occurrences of variables not in α and not in β, respectively. In case of a variable set $\mathbf{X}$, the (monotone) expression ϕ is necessarily in one variable.

Example 9. Let the VO-type $\theta = (Vars(R)^*Vars(S)^*)^*$ of q_{eq} of Example 3. The variable order $\mathbf{vo}(\theta, \phi_{eq})$ is obtained as follows. We first partition ϕ_{eq} in $\phi_1 = x_1y_1 + x_1y_2$ and $\phi_2 = x_2y_3 + x_3y_3$, each typed by $Vars(R)^*Vars(S)^*$. For

ϕ_1, we obtain $x_1 + x_1$ with type $Vars(R)^*$, and $y_1 + y_2$ with type $Vars(S)^*$, then $x_1 + x_1$ with type $Vars(R)$ and y_1 and y_2 with type $Vars(S)$, and finally the variable order $x_1 y_1 y_2$. We proceed similarly with ϕ_2 and obtain $x_2 x_3 y_3$. We concatenate the two orders and return $x_1 y_1 y_2 x_2 x_3 y_3$. □

Lemma 2. *For any query q and database D, **vo**(**type**$(q), \phi_{q,D})$ is a variable order, an instance of **type**(q), and can be computed in time $O(|\phi_{q,D}| \log^2 |\phi_{q,D}|)$.*

In case of hierarchical queries, the outcome of **vo** is a good variable order.

Lemma 3. *For any hierarchical query q and database D, $\pi =$ **vo**(**type**(q),$\phi_{q,D}$) is a good variable order for $\phi_{q,D}$ and the OBDD $(\phi_{q,D}, \pi)$ can be computed in time $O(|Vars(\phi_{q,D})|)$.*

Theorem 1 follows immediately from Lemmata 2 and 3.

4 IHQ$^{\neq}$ Queries

We extend the PTIME result of Theorem 1 to the strictly more expressive IHQ$^{\neq}$. We consider IHQ$^{\neq}$ queries Q_n with n hierarchical subqueries. We assume these subqueries ordered as in the tree representation of Q_n and denote by v_i the number of variables occurring in both $\phi_{Q_n,D}$ and the relation produced by computing the hierarchical subquery i on a database D $(1 \leq i \leq n)$. Then,

Theorem 2. *For any IHQ$^{\neq}$ query Q_n and database D there is a good variable order π for $\phi_{Q_n,D}$. In particular, π can be computed in time $O(|\phi_{Q_n,D}| \log^2 |\phi_{Q_n,D}|)$ and, given π, the OBDD $(\phi_{Q_n,D}, \pi)$ can be computed in time $O(|\phi_{Q_n,D}| \cdot (\sum_{i=1}^{n-1} (\prod_{j=1}^{i} v_j) v_i + \prod_{i=1}^{n} v_i))$.*

The above time complexity for OBDD construction can be exponential in the size of the fixed query q.

Example 10. Consider the database of Fig. 1, the query q_{neq} :- $R(A, B), S(C, D)$, $A \neq C$, and the associated expression ϕ_{neq} of Example 3. Assume R and S are partitioned according to the A-values. The R-partition for a_i is paired with all S-partitions but for a_i. The factored form of ϕ_{neq} makes the relationship between the variables of partitions explicit (Example 3).

Fig. 3 shows the OBDD (ϕ_{neq}, π_2). Let $\pi_2 = U \circ L$, where $U = x_1 \ldots x_4$ and $L = y_1 \ldots y_4$. We partition horizontally this OBDD in the upper part for U and the lower part for L. Any edge crossing the border from U to L points to a subgraph that represents a possibly partial sum of y's and is thus representable linearly in the number of y's. Less obvious is that the number of these subgraphs is at most quadratic in the number of x's (see Lemma 4 below). In short, this is because by setting to true at most two variables from different R-partitions, we reduce ϕ_{neq} to a sum of some y's. For instance, we reach the leftmost node y_1 by any of the assignments $x_1 = x_2 = 1$, $x_1 = x_3 = 1$, and $x_4 = 1$, and each of these cases covers all possible (exponentially many) assignments for the remaining variables. It turns out that π_2 is a good variable order for ϕ_{neq}. □

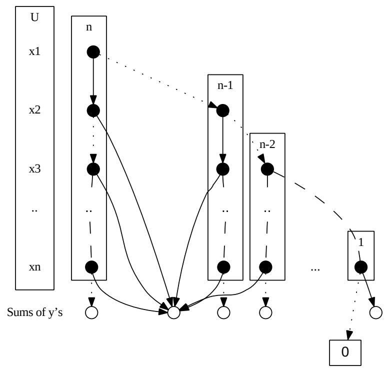


Fig. 7. Partial OBDD of quadratic size used in Lemma 4.

Like for hierarchical queries, we can derive VO-types for IHQ$^{\neq}$ queries and good variable orders using the functions **type** and **vo** described in Section 3.

We next discuss the base case of Theorem 2 with one inequality join on arbitrary relations, a generalization of q_{neq} from Example 10.

Lemma 4. *Let q :- $R(A_1, \dots, A_k), S(B_1, \dots, B_l), A_i \neq B_j$ for some $1 \leq i \leq k, 1 \leq j \leq l$, and database D. Then, π = **vo(type**(q),$\phi_{q,D}$) is a good variable order for $\phi_{q,D}$ and the OBDD $(\phi_{q,D}, \pi)$ can be computed in time $O(|\phi_{q,D}| \cdot (|Vars(R)|^2 + |Vars(R)| \cdot |Vars(S)|))$.*

Proof. Let $Vars(R) = \mathbf{X} = \{x_1, \dots, x_n\}$ and $Vars(S) = \mathbf{Y} = \{y_1, \dots, y_m\}$. The VO-type for q is $\mathbf{X}^*\mathbf{Y}^*$ and the variable order π =**vo**(**type**(q),$\phi_{q,D}$)= $U \circ L$, where $U = x_1 \dots x_n$ and $L = y_1 \dots y_m$. We partition the OBDD $(\phi_{q,D}, \pi)$ into the upper part for U and the lower part for L.

We show that (1) the number of nodes in the upper part is at most quadratic in n and (2) the number of nodes in the lower part is at most linear in $n \cdot m$.

(1) A variable of $\mathbf{X}$, which is associated in R with an A_i-value a, is paired in $\phi_{q,D}$ with all variables of $\mathbf{Y}$ associated in S with a B_j value different from a. This also means that by setting to true at most two variables associated with different A_i-values, we reduce $\phi_{q,D}$ to a (possibly partial) sum of y's. Fig. 7 shows our OBDD under the assumption that there is one variable in $\mathbf{X}$ per distinct A_i-value. The number of nodes on path 1 is n, on path 2 is $n-1$, and on path n is 1. We thus have $n \cdot (n-1)/2$ nodes in the upper part, which can be computed as shown in the figure. In case there are several variables in $\mathbf{X}$ associated with the same A_i-values, then they behave like one variable. If any one of them is set to true, then the remaining ones become redundant ($x_1 + \dots x_p = 1$ if at least one variable in the sum is 1); see the case of x_2 and x_3 in Fig. 3.

(2) Any edge crossing the border from the upper to the lower part points to a subgraph that represents a (possibly partial) sum of y's and thus linearly representable. This is because the expression $\phi_{q,D}$ is a bipartite monotone 2-DNF over $\mathbf{X}$ and $\mathbf{Y}$, and by crossing the border all variables of $\mathbf{X}$ are either set or irrelevant. The sum of all y's is reached from all upper part nodes having two variables of $\mathbf{X}$ set to true (depicted in Fig. 7 as the sum with most incoming

edges). Regarding the partial sums, there is one such sum for each of the n clusters, namely when all or all but one variable in $\mathbf{X}$ are set to false (In case $n < m$ some of these sums are equal).

We create the quadratic-size OBDD as follows. We first choose an arbitrary order of $\mathbf{X}$-variables followed by an arbitrary order of $\mathbf{Y}$-variables. For the construction of each node, we have a variable v and an expression ϕ (initially, $\phi = \phi_{q,D}$). We compute $\phi|_v$ and $\phi|_{\bar{v}}$ in two scans over ϕ. To detect that two expressions without $\mathbf{X}$-variables are the same, we only need to check in time linear in $|\phi_{q,D}|$ that they were created by setting to true two $\mathbf{X}$-variables corresponding to different A_i-partitions. $\square$

We next sketch the idea behind the proof of the general case of Theorem 2. We first simplify the input IHQ$^{\neq}$ query Q_n based on the observation that the hierarchical subqueries can be materialized to tuple-independent probabilistic relations. A good variable order π for $\phi_{Q_n,D}$ can then be obtained by concatenating the variables of the materialized relations, as given by the function **vo**.

If the tree of the simplified Q_n has under three leaves (corresponding to materialized hierarchical subqueries), then Theorem 1 or Lemma 4 applies. We otherwise construct the OBDD $(\phi_{Q_n,D}, \pi)$ by incrementally removing from Q_n hierarchical subqueries in the left-to-right order of their leaves in the tree. On removal, we create an OBDD fragment whose structure follows that of Fig.7.

Let H_1 and H_2 be the subquery to remove and a subquery that has an inequality with H_1, respectively. Let $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_l\}$ be the sets of (independent) expressions that occur in $\phi_{Q_n,D}$ and are associated with the tuples of the materialized H_1 and H_2, respectively.

Due to the inequality joins between H_2 and H_1 on one hand, and of H_2 and some of the remaining subqueries on the other hand, the expression $\phi_{Q_n,D}$ can be factored as $y_1 f(y_1) g(y_1) + \ldots + y_l f(y_l) g(y_l)$, where for any expression y_j, the function g is a sum of x_i's and the function f defines the cofactors of $y_j g(y_j)$ in $\phi_{Q_n,D}$. We can now apply the OBDD construction from the proof of Lemma 4, where we replace y_j by $y_j f(y_j)$. After constructing the OBDD fragment for the variables $\{x_1, \ldots, x_k\}$, we continue with the $O(k)$ expressions representing sums S_y of some $y_j f(y_j)$ for $1 \leq j \leq l$. We consider each such sum in separation and proceed by removing the next hierarchical subquery H. Each expression S_y can be factored according to the expressions of the materialized H and of a further materialized subquery sharing an inequality with H (if any), because their co-occurrence in the same clauses is not influenced by the fact that some $y_j f(y_j)$ are missing - there is no join between H_1 and H and hence all their dependencies are through some other subqueries.

By induction, the upper bound on the OBDD construction time is $O(|\phi_{Q_n,D}| \cdot (k^2 + k * \mathit{Rest}))$, where Rest is an upper bound for the size of sums S_y.

5 Hard Conjunctive Queries

We next discuss the case of general intractable conjunctive queries on restricted databases. The tractable classes of hierarchical queries and of queries with expressions $\phi_{q,D}$ of bounded treewidth are orthogonal, because the expressions $\phi_{q,D}$ do not admit in general bounded treewidth for hierarchical queries. Intuitively, this is because eq-joins lead in general to expressions $\phi_{q,D}$ consisting of clauses that pair an unbounded number of variables. We recall that the approach based on safe plans [5] cannot accommodate both aforementioned classes [6].

The #P-hard conjunctive queries have subqueries of the form [5]

$$R(\ldots, X, \ldots), S(\ldots, X, \ldots, Y, \ldots), T(\ldots, Y, \ldots).$$

Such queries are not hierarchical because the query variables X and Y have a common subgoal S and further distinct subgoals: R for X and T for Y. Intuitively, these queries are hard because they allow for arbitrary monotone DNF expressions (S can be constructed so as to allow arbitrary combinations of tuples of R and T), and some of them only admit exponential-size OBDDs [11].

Example 11. The expression $\phi_{q,D}$ for the hard query q :- $R(X,Z), S(X,Y), T(Y)$ and the database of Figure 1 is $x_1y_1z_1 + x_1y_2z_2 + (x_2 + x_3)y_3z_1$. All clauses are transitively dependent on each other and do not adhere to the regular factored-form pattern as in the case of hierarchical queries. □

Bounded Pathwidth. Our approach can benefit from existing significant work on tractable OBDD construction for Boolean expressions of bounded pathwidth or treewidth, e.g.,[12, 8, 9]. We use here the notion of pathwidth of the graph constructed from DNF formulas, where the nodes are variables and two nodes are directly connected if their variables occur in the same clause. The pathwidth of a graph measures how close the graph is to a path.

Definition 6 ([19]). *A path decomposition of a graph $G = (V, E)$ is a pair of path P with node set I and edge set F, and a family $L = \{L_i \mid i \in I\}$ of subsets of V such that: (1) $\bigcup_{i \in I} L_i = V$; (2) $\forall (v, w) \in E, \exists i \in I : \{v, w\} \subseteq L_i$; (3) $\forall i, j, k \in I$ if j is on the path from i to k in P, then $L_i \cap L_k \subseteq L_j$. The width of a path decomposition is $\max_{i \in I} |L_i| - 1$ and the pathwidth of a graph is the minimum width over all its possible path decompositions.*

The connection between pathwidth and treewidth follows by $pathwidth(G) = O(treewidth(G) \cdot \log n)$ for a graph G with n nodes. Using an argument similar to Theorem 2.1 of [9], we have that

Theorem 3. *For any query q and database D with $\phi_{q,D}$ of n variables and pathwidth p, $\phi_{q,D}$ has an OBDD of size $O(n2^p)$.*

In case p is bounded, $\phi_{q,D}$ admits a good variable order. The proof of Theorem 2.1 in [9] gives such an order: Let a path decomposition (P, L) for the graph of $\phi_{q,D}$ and define *First* and *Last* over the variables of $\phi_{q,D}$: $First(x) = \min\{n \in P \mid x \in L(n)\}$ and $Last(x) = \max\{n \in P \mid x \in L(n)\}$. A good variable order is the increasing lexicographic order of variables according to $(First(\cdot), Last(\cdot))$.

FD-induced Hierarchical Queries. We shortly mention a further important case of restricted databases that can ensure tractability of non-hierarchical queries. The idea is that under functional dependencies (FDs), non-hierarchical queries can sometimes admit equivalent hierarchical queries, and thus PTIME evaluation. Such equivalent hierarchical queries can be obtained by chasing the non-hierarchical query using FDs. Follow-up work [17] discusses this case in detail and shows that the conjunctive subqueries of most of the 22 TPC-H queries admit equivalent hierarchical rewritings under the TPC-H FDs.

References

1. L. Antova, T. Jansen, C. Koch, and D. Olteanu. "Fast and Simple Relational Processing of Uncertain Data". In *Proc. ICDE*, 2008.
2. O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. "ULDBs: Databases with Uncertainty and Lineage". In *Proc. VLDB*, 2006.
3. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2), 1992.
5. N. Dalvi and D. Suciu. "Efficient query evaluation on probabilistic databases". *VLDB Journal*, **16**(4):523–544, 2007.
6. N. Dalvi and D. Suciu. "Management of Probabilistic Data: Foundations and Challenges". In *Proc. PODS*, 2007.
7. N. Dalvi and D. Suciu. "The Dichotomy of Conjunctive Queries on Probabilistic Structures". In *Proc. PODS*, 2007.
8. A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4), 2001.
9. A. Ferrara, G. Pan, and M. Y. Vardi. Treewidth in verification: Local vs. global. In *Proc. LPAR*, 2005.
10. T. J. Green and V. Tannen. "Models for Incomplete and Probabilistic Information". In *Proc. IIDB*, 2006.
11. K. Hayase and H. Imai. OBDDs of a monotone function and of its prime implicants. In *Algorithms and Computation*, 1996.
12. J. Huang and A. Darwiche. Using DPLL for efficient OBDD construction. In *Revised Selected Papers of SAT 2004*, 2005.
13. T. Imielinski and W. Lipski. "Incomplete information in relational databases". *Journal of ACM*, **31**(4):761–791, 1984.
14. C. Koch and D. Olteanu. "Conditioning Probabilistic Databases". *JDMR (formerly Proc. VLDB)*, 1, 2008.
15. M. S. Lam, J. Whaley, V. B. Livshits, M. C. Martin, D. Avots, M. Carbin, and C. Unkel. Context-sensitive program analysis as database queries. In *PODS*, 2005.
16. C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag, 1998.
17. D. Olteanu, J. Huang, and C. Koch. "Lazy versus Eager Query Plans for Tuple-Independent Probabilistic Databases". Technical report, Oxford University, 2008.
18. L. D. Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proc. IJCAI*, 2007.
19. N. Robertson and P. Seymour. Graph minors. ii. algorithmic aspects of treewidth. *J. of Algorithms*, 7:309–322, 1986.
20. P. Sen and A. Deshpande. "Representing and Querying Correlated Tuples in Probabilistic Databases". In *Proc. ICDE*, 2007.