On Efficiently Evaluating Inequality Queries on Probabilistic Databases and Counting Vertex Covers

Rasmus Wissmann and Dan Olteanu

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK
{rasmus.wissmann, dan.olteanu}@comlab.ox.ac.uk

ABSTRACT

The problem of efficient evaluation of queries on probabilistic databases has received a great attention in recent years. In this paper, we introduce a class of queries with inequalities ($<, \leq$) and self-joins that admit tractable evaluation, in the context where query evaluation is #P-hard in general. This class strictly contains the class of inequality queries recently introduced by Olteanu and Huang.

We also show that particular tractable instances of our problem capture previously-open problems of counting vertex covers in chain and convex bipartite graphs, for which we can now easily derive efficient algorithms.

Our approach to both query evaluation and counting vertex covers is based on a novel syntactical characterization of k-DNF formulas that capture the lineage of our tractable queries and that can be compiled in at most quadratic time into binary decision diagrams of size linear in the size of the formulas.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms

Algorithms, Languages, Management, Performance

Keywords

Query Processing, Decision Diagrams, Probabilistic Databases

1. INTRODUCTION

Recent years have seen increased interest in the theory and development of probabilistic databases [6, 1, 2, 26, 14, 27, 28]. Such databases contain data that is only true with a given probability, and applications thereof include scientific databases, data integration, data cleaning and handling of sensor data [6, 1, 2, 26, 14, 27, 28]. Several research projects currently aim at developping database management systems (or parts thereof) for probabilistic relational data, e.g., MystiQ [3], Trio [2], MayBMS [11] and SPROUT [21], Bayestore [29], and MCDB [14].

One aspect vital to the success of all these systems is the availability of efficient query evaluation algorithms, in a context where exact query evaluation is #P-hard for conjunctive queries [7]. It is therefore commonly agreed that developping scalable query evaluation techniques has high priority. In the last five years, the map of achievements has accommodated both exact and approximate techniques. In case of approximate techniques, FPRAS (fully polynomial randomized approximation scheme) algorithms [15], which can compute an approximate answer confidence within a given allowed error with high probability, have been adapted to probability computation [23, 14, 11]. In case of exact techniques, the goals are to find classes of tractable queries (wrt data complexity) [5, 19, 20] and develop scalable evaluation strategies using relational query plans [6, 21].

One of the contributions of this paper fits into the latter category: We define tractable queries with inequalities ($<, \leq$) on tuple-independent probabilistic databases. This adds to the few known theoretical results concerning tractability of query evaluation on tuple-independent probabilistic databases, of which the dichotomy of conjunctive queries [5] and the trichotomy of "having" queries [24] are perhaps the most prominent examples. For conjunctive queries with inequalities ($<$), recent work [20] introduced a tractable class and presented a scalable secondary-storage evaluation algorithm that can be easily incorporated into existing relational query engines, such as that of PostgreSQL [20]. As we show later, that query class is strictly included in the class defined in this paper, while at the same time we obtain better bounds for the evaluation time.

The contributions of this paper are as follows:

- We introduce the class of acyclic one-out inequality queries and show that such queries can be evaluated in polynomial time (wrt data complexity) on tuple-independent probabilistic databases. This query class strictly includes the known tractable class of inequality queries [20], and, in contrast to the latter, it allows for self-joins, $\leq$ inequalities, and less structural constraints on the query inequality graph.
- We present a natural connection between two tractable inequality queries and bipartite chain and convex graphs. Using the tractability results developed in this paper,

we are able to show that counting the number of vertex covers or of independent sets can be done in polynomial time on such graphs.

- Both problems of query evaluation and counting essentially rely on a novel efficient compilation technique of so-called multi-chain k-DNF and convex 2-DNF formulas into binary decision diagrams, whose sizes are bounded by the number of literals in the formulas. The family of multi-chain k-DNFs captures the lineage of acyclic one-out queries. Convex 2-DNFs, whose graph representations form the class of convex bipartite graphs, precisely capture the lineage of a so-called single-guard query.

A long version of this paper is available from the web page of the SPROUT research project at Oxford [30].

2. PRELIMINARIES

2.1 Tuple-independent Probabilistic Databases

Let $\mathbf{X}$ be a finite set of (independent) Boolean random variables. *A tuple-independent probabilistic table* R is a relation of schema (A, V, P) with functional dependencies $A \to VP$, $V \to A$. The list A represents *data* columns, as in standard tables, whereas the values in the column V are variables from $\mathbf{X}$ and the values in the column P are numbers in $(0, 1]$ that represent the probabilities of the corresponding variables being true. A probabilistic database D is a set of probabilistic tables.

A probabilistic database represents a set of instance databases, or possible worlds, with one possible world for each total valuation of variables from $\mathbf{X}$. Under a total valuation θ, the instance of each probabilistic table R is the set of tuples (a) such that $(a, x, p) \in R$ and $\theta(x)$ is true. The probability of that world is the probability of the chosen total valuation f:

$$Pr[\theta] = \Big(\prod_{x\in\mathbf{X}:\theta(x)\text{ true}} Pr[x]\Big) \cdot \Big(\prod_{x\in\mathbf{X}:\theta(x)\text{ false}} Pr[\neg x]\Big).$$

2.2 Inequality Queries

We consider queries of the form

$$Q(A) : -R_1(A_1), \ldots, R_k(A_k), \phi(A_1, \ldots, A_k),$$

where the subgoals $R_1, \ldots, R_k$ do not necessarily represent disjoint relations, $A_1, \ldots, A_k$ are disjoint sets of query variables, $A \subseteq A_1 \cup \ldots \cup A_k$, and $\phi(A_1, \ldots, A_k)$ is a conjunction of inequalities $(<, \leq)$ over query variables. Without loss of generality, we only consider queries where the conjunction of inequalities is satisfiable and has no "local" inequalities that only involve query variables from the same subgoal and possibly constants. Satisfiability of conjunctions of inequalities can be checked efficiently [13]; inequalities local to one subgoal R can be easily resolved by dropping those tuples from relation R that do not satisfy these inequalities.

The query structure can be visualized using *query graphs*:

DEFINITION 2.1 (QUERY GRAPH). *The query graph of a query $Q(A) : -R_1(A_1), \ldots, R_k(A_k), \phi(A_1, \ldots, A_k)$ is the directed graph $G_Q = (V_Q, E_Q, R_Q)$, where V_Q is the set of query variables involved in inequalities, $(X_i, X_j) \in E_Q$ if $\phi(A_1, \ldots, A_k)$ contains the inequality $X_i \odot X_j$ for $\odot \in \{<, \leq\}$, and $R_Q = \{A_1 \cap V_Q, ..., A_k \cap V_Q\}$.*

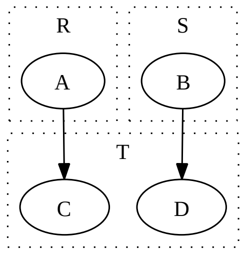

Figure 1: Query graph for the query of Example 2.2. A, B, C, D are query nodes, R, S, T are relation nodes.

R	A	V
	1	r_1
	2	r_2
	3	r_3

S	B	V
	1	s_1
	2	s_2
	3	s_3

T	C	D	V
	2	4	t_1
	3	3	t_2
	4	2	t_3

Figure 2: Tuple-independent probabilistic database (the probability columns P are not shown).

The sets in R_Q describe which query variables occur in the same subgoal. Graphically, they can be visualized by *relation nodes*, which are dotted boxes around query variable nodes.

EXAMPLE 2.2. The query graph for the Boolean query

$$Q : -R(A), S(B), T(C, D), A < C, B < D.$$

is shown in Figure 1. The sets of nodes, edges and relation nodes are $V_Q = \{A, B, C, D\}, E_Q = \{(A, C), (B, D)\}$ and $R_Q = \{\{A\}, \{B\}, \{C, D\}\}$. □

Conceptually, queries are evaluated in each world. Given a query Q and a probabilistic database D, the probability of a distinct answer tuple t is the probability of t being in the result of Q in the worlds of D, or equivalently,

$$\Pr[t \in Q(D)] = \sum_{\theta:\ t\in Q(D)\text{ in world }\theta} \Pr[\theta].$$

The evaluation of queries on probabilistic databases follows the standard semantics, where the columns for variables and probabilities are copied along in the answer tuples. These columns store relationally a DNF formula over Boolean random variables, which is commonly called *lineage*.

We denote the lineage of t by $\phi_{t,Q,D}$ (or ϕ_t if Q and D are clear). If Q is Boolean, we write ϕ_Q as a shorthand for $\phi_{(true),Q}$.

EXAMPLE 2.3. The answer of the Boolean query in Figure 1 on the database shown in Figure 2 is given below (left). The table obtained by the projection on the columns for random variables represents the relational encoding of the query lineage ϕ (right).

Q		V_R	V_S	V_T
		r_1	s_1	t_1
		r_1	s_1	t_2
		r_1	s_1	t_3
		r_1	s_2	t_1
		r_1	s_2	t_2
		r_1	s_3	t_1
		r_2	s_1	t_2
		r_2	s_1	t_3
		r_2	s_2	t_2
		r_3	s_1	t_3

$$\begin{aligned}\phi = \; & r_1s_1t_1+\\ & r_1s_1t_2+\\ & r_1s_1t_3+\\ & r_1s_2t_1+\\ & r_1s_2t_2+\\ & r_1s_3t_1+\\ & r_2s_1t_2+\\ & r_2s_1t_3+\\ & r_2s_2t_2+\\ & r_3s_1t_3.\end{aligned}$$

In an easier to read factored form, the lineage ϕ is:

$$\begin{aligned}\phi \quad = \quad & r_1\{s_1\,[t_1+t_2+t_3]+s_2\,[t_1+t_2]+s_3\,[t_1]\}\\ & +r_2\{s_1\,[t_2+t_3]+s_2\,[t_2]\}+r_3\{s_1\,[t_3]\}.\end{aligned}$$

Intuitively, the answer to Q would be true, if the answer lineage would be a tautology. Otherwise, it is only "partially" true, and this can be quantified by means of probabilities, as stated next. □

The following result is folklore.

PROPOSITION 2.4. *For any query Q, probabilistic database D, and a distinct tuple t in $Q(D)$,* $\Pr[t \in Q(D)] = \Pr[\phi_t]$.

Computing $\Pr[\phi_{t,Q,D}]$ is #P-complete in general and one goal of this paper is to define and study classes of queries for which this computation can be done efficiently.

Without loss of generality, we only consider Boolean queries in the sequel, because a non-Boolean query $Q(A)$ can be answered by evaluating repeatedly Boolean queries $Q[(\mathtt{a})/A]$ obtained by substituting some constant tuple $(\mathtt{a})$ for A.

We further introduce a few necessary tools.

The *size* of a formula ϕ, denoted by $|\phi|$, is the number of its literals.

DEFINITION 2.5. *Given two DNFs ϕ and ϕ', ϕ is* syntactically included *in ϕ', denoted by $\phi \subseteq \phi'$, if $c \in \phi'$ for every clause $c \in \phi$.*

DEFINITION 2.6. *Let a DNF ϕ and variables $x_1 \cdots x_l$ in ϕ. The* co-factor *of $(x_1 \cdots x_l)$ in ϕ, denoted by $f(x_1 \cdots x_l)$, is a DNF such that $\phi = (x_1 \cdots x_l)f(x_1 \cdots x_l) + \phi'$, and the DNF ϕ' does not contain clauses with all the variables $x_1 \cdots x_l$.*

2.3 Binary Decision Diagrams

Binary decision diagrams (BDDs) are commonly used to represent compactly large Boolean expressions [17].

The idea behind BDDs is to decompose Boolean formulas using variable elimination and possibly identify and share common subexpressions.

DEFINITION 2.7. *Given a formula ϕ and one of its variables x, its Shannon expansion is*

$$x \cdot \phi|_x + \; \bar{x} \cdot \phi|_{\bar{x}},$$

where $\phi|_x$ and $\phi|_{\bar{x}}$ are ϕ with x set to true and false, respectively. The expansion is exhaustive *if $\phi|_x$ and $\phi|_{\bar{x}}$ are literals.*

It is easy to see that any formula is equivalent to its Shannon expansion and that it can be brought into exhaustive Shannon expansion form by repeatedly applying the variable elimination step until no more such steps can be applied. Each decomposition step creates two subformulas and the variable elimination orders may differ for different subformulas.

BDDs can be seen as graphical representations of formulas in exhaustive Shannon expansion: They are directed acyclic graphs with two terminal nodes representing the constants 0 (false) and 1 (true), and non-terminal nodes representing the eliminated 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 identification and sharing of common subformulas is what can make BDDs more compact than the Shannon expansion form of Boolean formulas: a node n is redundant if both its outgoing edges point to the same node, or if there is a node for the same variable and with the same children as n.

EXAMPLE 2.8. Figure 3 gives a BDD for the DNF formula of Example 2.3. The terminal node 0 and its ingoing dotted edges are omitted to avoid clutter, but they can be easily derived: Each node without both outgoing edges shown has a dotted edge to node 0.

We show next several variable elimination steps that construct the Shannon expansion represented the the three uppermost nodes in the BDD. We first eliminate r_1 and obtain

$$\begin{aligned}\phi = r_1\phi|_{r_1} + \overline{r_1}\phi|_{\overline{r_1}},&\text{ where}\\ \phi|_{r_1} = s_1\,[t_1+t_2+t_3]+s_2\,[t_1+t_2]&+s_3\,[t_1]\\ +r_2\{s_1\,[t_2+t_3]+s_2\,[t_2]\}&+r_3\{s_1\,[t_3]\};\\ \phi|_{\overline{r_1}} = r_2\{s_1\,[t_2+t_3]+s_2\,[t_2]\}&+r_3\{s_1\,[t_3]\};\end{aligned}$$

Obviously, $\phi|_{r_1}$ can be simplified to a subset of its clauses: $s_1\,[t_1+t_2+t_3]+s_2\,[t_1+t_2]+s_3\,[t_1]$.

We next eliminate variable s_1 in $\phi|_{r_1}$ and obtain

$$\begin{aligned}\phi|_{r_1} &= s_1\phi|_{r_1s_1} + \overline{s_1}\phi|_{r_1\overline{s_1}},\text{ where}\\ \phi|_{r_1s_1} &= [t_1+t_2+t_3]+s_2\,[t_1+t_2]+s_3\,[t_1]\\ \phi|_{r_1\overline{s_1}} &= s_2\,[t_1+t_2]+s_3\,[t_1]\end{aligned}$$

Again, $\phi|_{r_1s_1}$ can be simplified to $t_1+t_2+t_3$. We can now eliminate t_1 and obtain

$$\begin{aligned}\phi|_{r_1s_1} &= t_1\phi|_{r_1s_1t_1} + \overline{t_1}\phi|_{r_1s_1\overline{t_1}},\text{ where}\\ \phi|_{r_1s_1t_1} &= 1 \qquad \text{and } \phi|_{r_1s_1\overline{t_1}} = t_2+t_3\end{aligned}$$

□

REMARK 2.9. The probability of a BDD representing a formula over Boolean random variables can be computed in time linear in its size using the fact that the branches of any node represent mutually exclussive expressions, e.g., [19]. The probability of any internal node n is the sum of the probabilities of their children weighted by the probabilities of the corresponding assignments of the decision variable at

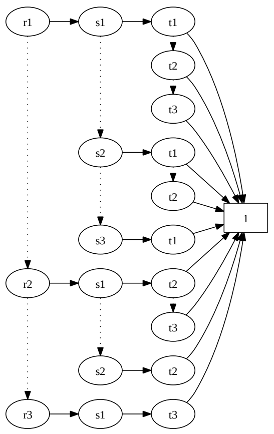

Figure 3: BDD for Example 2.3

that node. The probability of the terminal nodes is given by their label (1 or 0).

The same procedure is applicable to model counting, i.e., to counting the number of satisfying assignments of Boolean formulas represented as BDDs with the following addition. If we set the probability of each random variable to 1/2, we then obtain the probability $c \cdot \frac{1}{2^n}$, where n is the number of variables and c is the number of satisfying assignments of the input formula. □

3. BDDS FOR MULTI-CHAIN K-DNFS

In this section we define a class of k-DNFs that admit tractable model counting and probability computation. Their tractability is inherited from BDDs, into which they can be efficiently compiled.

Definition 3.1 (Multipartite k-DNF). *A formula is in k-DNF if it is in disjunctive normal form and each clause has k literals. A k-DNF is* multipartite *over the list of variable sets $[X_1, \ldots, X_k]$ if these sets are pairwise disjoint or equal and the l-literal in each clause is from X_l for $1 \leq l \leq k$.*

The lineage of conjunctive queries on tuple-independent databases can be expressed as multipartite k-DNFs, where each set X_i consists of the random variables of a tuple-independent relation. In case of 2-DNFs, they match the separation of nodes into disjoint sets of bipartite graphs.

A special form of multipartite k-DNFs, called multi-chain, is of particular interest here.

Definition 3.2. *A multipartite k-DNF over the list of variable sets $[X_1, \ldots, X_k]$ is* multi-chain *if there are orders $<_1, \ldots, <_k$ over the variables of each of its sets such that*

$$\forall 1 \leq j \leq l \leq k, x_j \in X_j, y_l \in X_l, x_l <_l y_l :$$
$$f(x_1 \cdots x_l) \supseteq f(x_1 \cdots x_{l-1} y_l) \text{ or } f(x_1 \cdots x_l) = 0.$$

Example 3.3. The formula $\phi = x_1y_1 + x_1y_2 + x_1y_3 + x_2y_2 + x_2y_3$ is a multipartite 2-DNF over disjoint variable sets $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2, y_3\}$. It is also multi-chain, since $f(x_1) = y_1 + y_2 + y_3 \supseteq y_2 + y_3 = f(x_2)$. □

The naming "multi-chain" comes from the fact that the graph of a multi-chain 2-DNF formula over disjoint variable sets is a bipartite *chain* graph. We discuss the connection to such graphs in more detail in Section 5.

The main result of this section is stated next.

Theorem 3.4. *Any multi-chain ϕ with orders $<_1, \ldots, <_k$ can be compiled in time $O(|\phi|^2)$ into a BDD of size bounded by $|\phi|$. If the variable sets of ϕ are disjoint, then the compilation time is $O(|\phi| \log |\phi|)$.*

The proof is based on the following observation: Multi-chain k-DNFs have the property that the elimination of the first variable in the order $<_1$ leads to a (not necessarily DNF) Shannon expansion formula of size linear in the size of the input formula. This property turns out to be crucial for the tractability of model counting. We exemplify this property with the formula ϕ of Example 3.3:

$$\begin{aligned}\phi &= x_1 \cdot \phi|_{x_1} + \overline{x_1} \cdot \phi|_{\overline{x_1}}\\ &= x_1 \cdot [y_1 + y_2 + y_3 + x_2y_2 + x_2y_3] + \overline{x_1} \cdot [x_2y_2 + x_2y_3]\\ &= x_1 \cdot [f(x_1) + x_2 \cdot f(x_2)] + \overline{x_1} \cdot x_2 \cdot f(x_2)\end{aligned}$$

The cofactors of x_1 and x_2 are here $f(x_1) = y_1 + y_2 + y_3$ and $f(x_2) = y_2 + y_3$. Since $f(x_1) \Rightarrow x_2 f(x_2)$, we have that $f(x_1) + x_2 f(x_2) = f(x_1)$ and $f(x_1) \supset f(x_2)$, and thus

$$\phi = x_1 f(x_1) + \overline{x_1} \cdot x_2 \cdot f(x_2).$$

The cofactors $f(x_1)$ and $f(x_2)$ are trivially multi-chain. The following lemma states that this property of cofactors holds even for multi-chains with $k > 2$. We could then recursively apply variable elimination to the obtained cofactors until we reach an exhaustive Shannon expansion.

Lemma 3.5. *Any multi-chain ϕ over variable sets $[X_1, \ldots, X_k]$ is equivalent to ϕ_1, where $n = |X_1|$ and*

$$\begin{aligned}\phi_i &= x_i f(x_i) + \overline{x_i} \phi_{i+1}, \forall 1 \leq i < n\\ \phi_n &= x_n f(x_n),\end{aligned}$$

The cofactors $f(x_1), \ldots, f(x_n)$ are multi-chain (k-1)-DNFs.

Proof. It is assumed that the variables in X_1 are ordered by $<_1$ such that the syntactical inclusion of Definition 3.2 holds.

By Definition 2.6 we have

$$\phi = x_1 f(x_1) + x_2 f(x_2) + \ldots + x_n f(x_n).$$

Since ϕ is multi-chain it holds $f(x_i) \supseteq f(x_j)$, $\forall 1 \leq i < j \leq n$, and thus $\neg f(x_i)$ implies $\neg f(x_j)$ for all $i < j$. This means

$$f(x_i) + x_{i+1} f(x_{i+1}) + \ldots + x_n f(x_n) = f(x_i)$$

for any choice of i. Thus,

$$\begin{aligned}\phi &= x_1 f(x_1) + \overline{x}_1 [x_2 f(x_2) + \ldots + x_n f(x_n)]\\ &= x_1 f(x_1) + \overline{x}_1 [x_2 f(x_2) + \overline{x}_2 [x_3 f(x_3) + \ldots + x_n f(x_n)]]\\ &= \ldots,\end{aligned}$$

which proves the expansion.

Now each cofactor $f(x_i)$, $1 \leq i \leq k$, is a multipartite (k-1)-DNF over variable sets $X_2, \ldots, X_k$. To show that it is multi-chain, denote the cofactors in $f(x_i)$ by f_i and let $2 \leq j \leq l \leq k, y_j \in X_j, z_l \in X_l, x_l <_l y_l$ be arbitrary. Then

$$\begin{gathered} f_i(y_2 \cdots y_l) \supseteq f_i(y_2 \cdots y_{l-1} z_l) \text{ or } f_i(y_2 \cdots y_l) = 0\\ \Leftrightarrow\\ f(x_i y_2 \cdots x_l) \supseteq f(x_i y_2 \cdots y_{l-1} z_l) \text{ or } f(x_i y_2 \cdots y_l) = 0,\end{gathered}$$

which is true because ϕ is multi-chain. Hence, each $f(x_i)$ is a multi-chain (k-1)-DNF. □

Lemma 3.6. *Any multi-chain ϕ can be expressed as an exhaustive Shannon expansion ϕ' with*

$$|\phi'| \leq |\phi| + |X_1| + \ldots + |X_k| \leq 2|\phi|.$$

Proof. Assume ϕ has variable sets $[X_1, \ldots, X_k]$. Lemma 3.5 shows how we can eliminate the variables of the first set X_1 and that all the cofactors of variables in this set are multi-chain $(k-1)$-DNFs. Thus, we can apply the factorization of Lemma 3.5 k times and obtain an exhaustive Shannon expansion ϕ'.

We inductively show the upper bound: ϕ is expressible as

$$x_1 f(x_1) + x_2 f(x_2) + \ldots + x_n f(x_n)$$

where the number of literals in each $x_i f(x_i)$ is smaller than the number of literals in the sum of all conjunctions of ϕ which start with x_i, because x_i appears only once (and all other literals appear in both). Thus,

$$|x_1 f(x_1) + x_2 f(x_2) + \ldots + x_n f(x_n)| \leq |\phi|.$$

After the application of Lemma 3.5, we obtain the equivalent formula ϕ_1, where for each variable in X_1 we add one negated literal. This gives

$$|\phi_1| \leq |\phi| + |X_1|.$$

Inductively, this leads to

$$|\phi'| \leq |\phi| + |X_1| + \ldots + |X_k| \leq 2|\phi|$$

for the exhaustive Shannon expansion ϕ'. ☐

An exhaustive expansion was shown earlier in this section for the formula of Example 3.3.

Lemma 3.7. *Any multi-chain ϕ can be brought into exhaustive Shannon expansion in $O(|\phi| \log |\phi|)$ if its variable sets are pairwise disjoint, or in $O(|\phi|^2)$ if some of its variable sets are equal.*

Proof. We first sort ϕ according to the orders $<_1, \ldots, <_k$, which can be done in $O(|\phi| \log |\phi|)$.

Consider first the case of pairwise disjoint variable sets. Thus, no variable can occur several times in the same clause and the cofactors of variables in X_i are $(k-i)$-DNFs.

We can then in one scan over ϕ rewrite it as

$$\phi = x_1 f(x_1) + \ldots + x_n f(x_n).$$

In a second scan, we compute again the cofactors of variables in X_2 within each cofactor of the variables in X_1. With $Y = X_2$ this gives

$$\begin{aligned}\phi \;=\; & x_1(y_1 f(x_1 y_1) + \ldots + y_m f(x_1 y_m)) \\ & + \ldots + x_n(y_1 f(x_n y_1) + \ldots + y_m f(x_n y_m)).\end{aligned}$$

This completes the expansion after k scans, i.e. in time $O(k|\phi|)$. In fact, we can do everything in one scan, if we exhaustively expand $f(x_1)$ before moving tp the next cofactors of variables in X_1.

If some sets of variables are identical, the same variable can occur multiple times in a clause. When computing the cofactor $f(x)$ for a variable x, we then replace all occurrences of x by 1 (true) in $f(x)$, and eliminate all clauses in $f(x)$ that are subsumed by clauses with occurrences of 1. This requires quadratic time in the number of clauses. Furthermore, we only compute cofactors for variables of ϕ that are in sets $X_2, \ldots, X_k$, thus skipping cofactors for the constant 1. ☐

```
MultiChain2BDD(multi-chain $k$-DNF $\phi$, $<_1, \ldots, <_k$) {

Assumption: $\phi$ ordered according to $<_1, \ldots, <_k$;

  // Keep nodes for variables set to true on current BDD path
  define nodes as array of $k+1$ BDD nodes;
  // Create BDD for the first clause
  nodes[k+1] = 1; // constant node 1
  foreach $i$ from $k$ downto 1 do
    nodes[i] = new  BDDnode($\phi$.clause(1).literal(i));
    nodes[i].high = nodes[i+1];
  endfor
  root = nodes[1];
  // Add BDD nodes for each clause
  foreach clause c in $\phi$ do
    // Find position where path continues
    d = FirstDifferentLiteral(c,c.lastClause());
    // Create new nodes for the last $k-d$ positions
    foreach $i$ from $k$ downto $d+1$ do
      nodes[i].low = 0; // constant node 0
      nodes[i] = new  BDDnode(c.literal(i));
      nodes[i].high = nodes[i+1];
    endfor
    // Create new node for position d and connect
    // the low edge from the previous node
    temp = nodes[d];
    nodes[d] = new  BDDnode(c.literal(d));
    nodes[d].high = nodes[d+1];
    temp.low = nodes[d];
  endfor
  // Connect the remaining low edges to 0
  foreach $i$ from 1 to $k$ do nodes[i].low = 0;
  return root;
}
```

Figure 4: The MultiChain2BDD algorithm.

Proof of Theorem 3.4. Formulas written in exhaustive Shannon expansion are nothing else than term notations for BDDs, where each variable occurs once positive and once negated in the formula; the positive and negative occurrences account for the solid and dotted edge, respectively, of the BDD node for that variable. For multi-chain k-DNFs with disjoint sets of variables, Theorem 3.4 now follows from Lemmas 3.5, 3.6, and 3.7. ☐

Figure 4 gives a more practical algorithm for the construction of the BDD. This algorithm works on relational encodings of multi-chain k-DNFs where its variables sets are sorted in the orders $<_1, \ldots, <_k$, and constructs a BDD in one scan over the input. Regarding memory footprint, the algorithm only needs to keep in main memory one array of $k+1$ nodes, the BDD nodes created during the scan of the formula. The time needed by the algorithm is $O(|\phi|)$.

We explain the algoritm with the running example illustrated in Figure 5. We are given the multi-chain 4-DNF formula from the left table. The BDD at the right is shaped similarly to the tabular representation of the formula to show its structural similarity; the BDD can be seen as a factorization of the formula, where r_1 occurs only once (it is factored out), within r_1, s_1 occurs only once, and when the clauses with s_1 are exhausted, s_1 is invalidated and we move to s_2 (see the dotted edge between the nodes with variables s_1 and s_2). Similar factorizations happen for the

1	2	3	4	
r_1	s_1	t_1	u_1	
r_1	s_1	t_1	u_2	
r_1	s_1	t_2	u_1	
r_1	s_1	t_2	u_2	
r_1	s_1	t_3	u_2	
r_1	s_2	t_2	u_1	$\leftarrow$
r_1	s_2	t_2	u_2	
r_1	s_2	t_3	u_2	
r_2	s_1	t_2	u_1	
⋮	⋮	⋮	⋮	

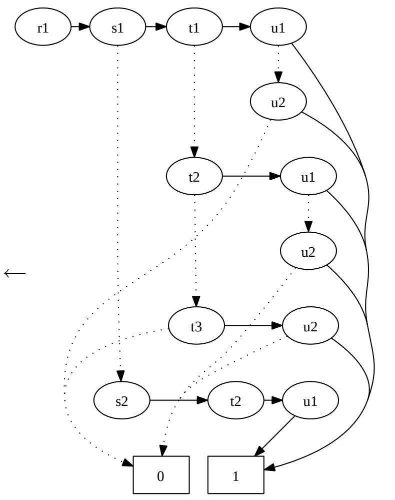

Figure 5: Running example for MultiChain2BDD. The left table shows a relational encoding of a multi-chain 4-DNF. At right, the BDD as constructed by the algorithm after seing the first six clauses.

other variables.

On reading the first clause, we construct the first line of BDD nodes that are connected by solid edges (the "high" field): The formula is true if the variables in each of these nodes are set to true. We then move to the second clause and identify the first literal different from the first clause (procedure `FirstDifferentLiteral`) in the variable at the 4th position (u_2 instead of u_1). We create a new BDD node for the new literal u_2 and connect it to the low edge of the previous node for u_1: The formula is true if r_1, s_1, t_1 are true and u_1 or u_2 are true. The compilation follows similarly until the sixth clause is added to the BDD, the result being the (partial) BDD in Figure 5.

In case some of the variable sets of the input k-DNF are equal, there may be clauses with the same variable occurring several times. In such cases, the algorithm does not create a valid BDD as there may be several nodes for the same variable on the same path. Nevertheless, the outcome of the algorithm can be turned into a BDD in one scan by removing such nodes.

Lemma 3.8. *Any multi-chain k-DNF ϕ with some of the variable sets equal, can be compiled into a BDD in time $O(|\phi|^2)$.*

Proof. Let $B(\phi)$ be the outcome of the algorithm MultiChain2BDD for ϕ. As explained above, to turn $B(\phi)$ into a BDD we need to remove illegal nodes along each of its paths. We show below that this can be done in time quadratic in the size of $B(\phi)$. Since this size is bounded by $O(|\phi|)$, and MultiChain2BDD needs linear scan of ϕ, we obtain the upper bound of $O(|\phi|^2)$.

We scan each path and keep track of all the variables of the nodes along that path. When we arrive at a node, we check whether its variable has been already encountered along the path. If this is not the case, we advance to the next node. Otherwise, we drop the node and only keep its positive or negative branch if the variable at the earlier node is reachable via its positive or negative edge, respectively. This test requires to compare the variable of each node with the variables of nodes up in the path, hence quadratic time in the path length. $\square$

Remark 3.9. Compiling formulas into BDDs is a well-investigated research field in AI with many fundamental contributions, e.g.,[17]. The problem of tractable compilation into propositional theories beyond the popular reduced ordered BDDs [4, 12] has received tremendous attention, for instance, the work of Darwiche on decomposable negation normal form and its variations [8, 9]. These approaches contrast to ours in that they bound the size of the constructed BDD by the treewidth of the input formula. Our formulas, however, can have unbounded treewidth. For instance, the 2-DNF formula over sets X_1 and X_2 that has a clause for each pair of variables from the two sets has unbounded treewidth. This 2-DNF formula is multi-chain. $\square$

4. ACYCLIC ONE-OUT QUERIES

We next define a class of tractable inequality queries called *acyclic one-out*.

Definition 4.1. *An inequality query is* acyclic*, if there is no cycle of relation nodes in its query graph. It is* one-out*, if for each relation, at most one of its nodes has outgoing edges in the query graph.*

Example 4.2. Figure 1 shows the graph of the acyclic one-out query $Q : -R(A), S(B), T(C, D), A < C, B < D$. If we would add the inequality $D < A$, then the query would be cyclic. If in addition we would add the inequality $C < B$, then the query would not be one-out, for there would be two outgoing edges from both query variables of T. $\square$

We obtain tractability of acyclic one-out queries by showing that the lineage of any query in this class is a multi-chain k-DNF, where the order of variable sets in the k-DNF follows any topological order of the corresponding relations in the query graph.

Lemma 4.3. *The lineage of any acyclic one-out query is a multi-chain k-DNF formula.*

Proof. W.l.o.g., Q can be written as

$$Q : -R_1\left(A_1^1, \ldots, A_{N_1}^1, B^1\right), \ldots, R_k\left(A_1^k, \ldots, A_{N_k}^k, B^k\right), \Lambda$$

where

1. $B^1, \ldots, B^k$ are the lists of query variables not participating in inequalities,
2. Λ is the conjunction of inequalities over query variables,
3. Only the nodes for $A_1^1, \ldots, A_1^k$ have outgoing edges in the query graph,
4. Each relation R_i is sorted ascendingly on the first column A_1^i, and
5. The order of subgoals follow a topological order obtained from the query graph, i.e., there is no outgoing edge from nodes of R_j to nodes of R_i for $1 \leq i < j \leq k$.

The third assumption can be made since Q is one-out and both Q and the relations $R_1,\dots,R_k$ can be trivially reorganized by moving the query variables and their corresponding attributes, respectively, as first in the subgoals and relations, respectively. The fifth assumption holds since Q is acyclic.

By construction, the query lineage ϕ is a multipartite k-DNF over the variable sets $[X_1,\dots,X_k]$, where X_i is the set of random variables occurring in R_i. We next show that it is also multi-chain.

We first introduce some notation. Each relation R_i, $1 \le i \le k$, can be written as

$$R_i = \left\{(v^i_{1,1},...,v^i_{1,N_i},b^i_1,x^i_1),...,(v^i_{n_i,1},...,v^i_{n_i,N_i},b^i_{n_i},x^i_{n_i})\right\}$$

where n_i is the size of R_i, $v^i_{j,r}$ and b^i_j are the data values of attributes A^i_r and B^i, respectively, and $x^i_j \in X_i$.

Let $1 \le i_1 \le n_1,\dots,1 \le i_k \le n_k$, $1 \le l \le k$, and $\Lambda = \Lambda_1 \wedge \Lambda_2$ where Λ_1 contains all inequalities over query variables from R_l.Thus, Λ_1 consists of inequalities of the form $A^{l'}_{m'} \odot A^l_m$ (ingoing) with $l' < l$ and $A^l_1 \odot A^{l''}_{m''}$ (outgoing) with $l < l''$, where $\odot \in \{<,\le\}$, $1 \le m \le n_l$, $1 \le m' \le n_{l'}$, $1 \le m'' \le n_{l''}$.

If the clause $x^1_{i_1}\cdots x^k_{i_k}$ is in ϕ, then the corresponding data values $(v^1_{i_1,1},\dots,v^1_{i_1,N_1}),\dots,(v^k_{i_k,1},\dots,v^k_{i_k,N_k})$ satisfy Λ.

Consider now an arbitrary $1 \le i'_l < i_l$ and $f(x^1_{i_1}\dots x^{l-1}_{i_{l-1}}x^l_{i'_l}) \neq \emptyset$. Since the data values for all attributes other than A^l_1 are not changed, Λ_2 is satisfied by $(v^1_{i_1,1},\dots,v^1_{i_1,N_1}),\dots,(v^l_{i'_l,1},\dots,v^l_{i'_l,N_l}),\dots,(v^k_{i_k},\dots,v^k_{i_k,N_k})$.

Since $f(x^1_{i_1}\dots x^{l-1}_{i_{l-1}}x^l_{i'_l}) \neq \emptyset$, the ingoing inequalities of Λ_1 are satisfied by $(v^1_{i_1,1},\dots,v^1_{i_1,N_1}),\dots,(v^l_{i'_l,1},\dots,v^l_{i'_l,N_l})$. Furthermore, we have

$$v^l_{i'_l,1} \le v^l_{i_l,1}$$

and thus the implication

$$v^l_{i_l,1} \odot v^{l'}_{i_{l'},m'} \Rightarrow v^l_{i'_l,1} \odot v^{l'}_{i_{l'},m'}$$

for any $\odot \in \{<,\le\}$, $l' \ge l, 1 \le m' \le N_{l'}$. This shows that the outgoing inequalities of Λ_1 are also satisfied. Since i_l and i'_l are arbitrary it follows that

$$f(x^1_{i_1}\dots x^{l-1}_{i_{l-1}}x^l_{i_l}) \subseteq f(x^1_{i_1}\dots x^{l-1}_{i_{l-1}}x^l_{i'_l})$$

for arbitrary $1 \le l \le k$, $1 \le i'_l < i_l \le n_l$ and $1 \le i_1 \le n_1,...,1 \le i_{l-1} \le n_{l-1}$ for which $f(x^1_{i_1}\dots x^{l-1}_{i_{l-1}}x^l_{i'_l}) \neq 0$. Thus, ϕ is multi-chain. $\square$

By Lemma 4.3 and Theorem 3.4 it then follows that

Theorem 4.4. *The data complexity of acyclic one-out queries is PTIME. In particular, the probability of the lineage ϕ of such queries can be computed in time $O(|\phi|^2)$ for queries with self-joins, and in time $O(|\phi|\log|\phi|)$ for queries without self-joins.*

Remark 4.5. The class of acyclic one-out queries strictly contains the *max-one* inequality queries without self-joins, which were previously shown to be tractable [20]. A query has the max-one property if at most one query variable per subgoal participates in inequalities. Max-one queries thus forbid not only that several graph nodes of the same relation have outgoing edges, but also that several graph nodes of the same relation have incoming edges. For instance, the query in the previous example is not max-one since the relation T has two nodes (C and D) with incoming edges. In addition, acyclic one-out queries can have self-joins and $\le$ operators, which are not allowed in max-one queries.

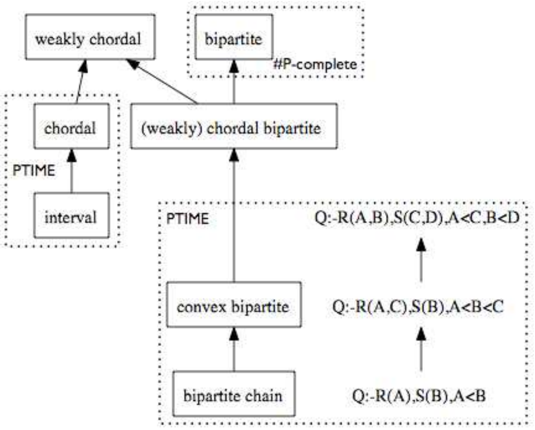

Figure 6: Complexity of #VC for various graphs.

Our present work also differs from [20] in that the technique of that work cannot be used to show tractability of acyclic one-out queries. The technique employed there compiles query lineage into ordered BDDs (OBDDs), whereas in the case of the present paper we drop the ordering constraint. Indeed, in general the BDDs for multi-chain k-DNFs can only be translated into OBDDs of exponential size. This argument is supported by the fact that BDDs can be exponentially more succinct that OBDDs [17].

The same earlier work of the authors [20] defined an extension of tractable inequality queries with equality joins. This extension considers so-called *efficient independent* queries, for which the distinct answer tuples are pairwise independent and their probability can be com- puted in polynomial time. It is then easy to see that such queries can be used at the place of subgoals in acyclic one-out inequality queries without affecting their tractability. $\square$

5. COUNTING VERTEX COVERS

Our tractability results on BDD construction for k-DNFs are applicable to counting the number of vertex covers (#VC) in chain and convex bipartite graphs. Remarkably, both chain and convex bipartite graphs are captured by the lineage of inequality queries on tuple-independent probabilistic databases.

We first recall necessary definitions.

Definition 5.1 ([25]). *A bipartite graph $G = (X,Y,E)$ is a* bipartite chain *graph if and only if for both X and Y the neighborhoods of the nodes can be ordered linearly w.r.t. (non-strict) inclusion.*

A bipartite graph $G = (X,Y,E)$ is convex *if there is an ordering $<$ of X or Y such that the neighborhoods of the vertices in the other set are consecutive in the ordering $<$ (i.e. they are intervals).*

Definition 5.2. *Let $G = (V,E)$ be an (undirected) graph. $S \subseteq V$ is a* vertex cover *of G, if for each edge $e \in E$ at least one of endpoints of e is in S.*

The main result of this section is given next.

THEOREM 5.3. *#VC is in $O(|E|^2)$ for any bipartite chain graph $G(X, Y, E)$.*

THEOREM 5.4. *#VC is in $O(|E|^2)$ for any convex bipartite graph $G(X, Y, E)$.*

To put these results in context, we note that counting vertex covers in general graphs is #P-complete [10]. By Proposition 5.5 it then follows that #VC(bipartite graphs) and #SAT(monotone bipartite 2-DNF) are #P-complete as well [22]. Our multi-chain 2-DNF formulas are thus necessarily restrictions of bipartite 2-DNFs.

PROPOSITION 5.5 ([22]). *The following problems are polynomial-time reducible to each other: #VC(graphs), #VC (bipartite graphs), and #SAT(monotone bipartite 2-DNF).*

Figure 6 gives further complexity results for #VC in classes of graphs. An arrow from class A to B means that A is strictly included in B. #VC is in PTIME for chordal graphs [16, 18] and #P-complete for bipartite graphs [22].

In the sequel, we prove Theorems 5.3 and 5.4. We first show the connection between #VC and query evaluation. This connection follows in two steps: Firstly, the equivalence between #VC and counting the number of satisfying assignments (#SAT) for DNFs, and secondly the connection between #SAT and query evaluation by Proposition 2.4.

The connection between vertex cover and SAT is as follows [22]: Given a bipartite graph $G = (X, Y, E)$, create a monotone bipartite 2-CNF ϕ with variables $x_i \in X$ and $y_j \in Y$ such that $(x_i \vee y_j) \in \phi$ if and only if $(x_i, y_j) \in E$. Then, $S \subseteq (X \cup Y)$ is a vertex cover of G exactly when the assignment $\sigma : (X \cup Y) \rightarrow \{0, 1\}$ with $\sigma(z) = 1 \Leftrightarrow z \in S$ is in SAT(ϕ).

Given ϕ, one can easily create a monotone bipartite 2-DNF ϕ', if we exchange conjunctions and disjunctions. Then, $\sigma \in$ SAT(ϕ) if and only if $\neg\sigma \notin$ SAT(ϕ') (i.e. $\neg\sigma$ is in the set NSAT(ϕ') of non-satisfying assignments of ϕ'). Since the number of all total truth assignment of $|X \cup Y|$ Boolean variables is $2^{|X \cup Y|}$, we can easily relate #SAT and #NSAT.

In the sequel, let $G(\phi)$ denote the bipartite graph associated with a monotone bipartite 2-DNF ϕ with variable sets X and Y. It is given by $G(\phi) = (X, Y, E)$ with $(x_i, y_j) \in E$ if and only if $x_i y_j \in \phi$.

EXAMPLE 5.6. Figure 7 shows the graph $G(\phi)$ for the 2-DNF

$$\phi = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3.$$

Example vertex covers are $S_1 = \{x_1, x_2, x_3\}$, $S_2 = \{x_1, x_3, y_1, y_2\}$ and $S_3 = \{x_1, x_2, x_3, y_1, y_2, y_3\}$. The corresponding non-satisfying assignments for ϕ are $\sigma_i : X \cup Y \rightarrow \{0, 1\}$, $i \in \{1, 2, 3\}$ with $\sigma(z) = 1$ iff $z \in S_i$.

In total, $G(\phi)$ has 18 distinct vertex covers. This implies that ϕ has 18 non-statisfying and $2^6 - 18 = 64 - 18 = 46$ satisfying assignments. □

Figure 6 also gives, next to graph classes, inequality queries, whose lineage graphs are chain or vertex.

LEMMA 5.7. *Let the query $Q : -R(A), S(B), A < B$. Then, the class of lineage graphs of Q is the class of bipartite chain graphs.*

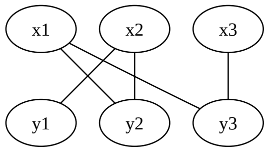

Figure 7: Bipartite graph for 2-DNF of Example 5.6.

PROOF. Let us denote by $N(x)$ the neighbourhood of a node x, i.e., all Y-nodes directly connected to x. Analogously, $N(x)$ for a variable x is the set of Y-variables that occur with x in clauses of the query lineage.

"$\subseteq$": Let ϕ be the query lineage, which is a bipartite 2-DNF over variable sets [X,Y]. Then, its graph is (X, Y, E) constructed as explained above. If the variables in R and S are ordered ascendingly by A and B respectively implies that $N(x_1) \supseteq N(x_2) \supseteq \ldots$ and $N(y_1) \subseteq N(y_2) \subseteq \ldots$. Thus, the lineage graph is a bipartite chain graph.

"$\supseteq$": Let $G = (X, Y, E)$ be a bipartite chain graph. Then there exists an order of X and Y such that $N(x_1) \supseteq N(x_2) \supseteq \ldots$ and $N(y_1) \supseteq N(y_2) \supseteq \ldots$. Now create a database D with relations $R(A)$ and $S(B)$ according to the following scheme: For each $x_i \in X$ add a tuple (i) with random variable x_i to R. For each $y_j \in Y$ add a tuple $(k + 0.1)$ with random variable y_j to S, where k is the maximum i such that $y_j \in N(x_i)$.

For arbitrary i, j, the clause $x_i y_j$ is in the query lineage ϕ if and only if $i < k + 0 - 1 = max_i(y_j \in N(x_i)) + 0.1$. Thus, $x_i y_j \in \phi$ if and only if $\exists k \geq i$ such that $y_j \in N(x_k)$. Since $N(x_1) \supseteq N(x_2) \supseteq \ldots$, this implies that $x_i y_j \in \phi$ if and only if $y_j \in N(x_i)$. Thus, an edge (i, j) is in the ϕ's graph if and only if (i, j) is in E. □

We are now ready to complete the proof of Theorem 5.3.

PROOF. Consider the multi-chain 2-DNF ϕ over the variable sets $[X, Y]$ corresponding to the bipatite chain graph $G = (X, Y, E)$. The orders $<_X$ and $<_Y$ can be computed in time quadratic in the number of the clauses of ϕ (i.e., $O(|E|^2)$) - this is necessary for computing the cofactor of each variable in X (or, equivalently, the neighbourhood of each node in X), and then finding the linear inclusion of these cofactors.

The DNF ϕ can be compiled into a BDD of linear size in quadratic time (see Theorem 3.4, Lemma 4.3, Proposition 5.5, and Lemma 5.7). □

For convex bipartite graphs, we have the following results that related them to the so-called single-guard query $Q : -R(A, C), S(B), A < C, C < B$ (where the query variables A and C *guard* B). We note that this query is cyclic.

LEMMA 5.8. *The class of lineage graphs of the single-guard query is the class of convex bipartite graphs.*

PROOF. "$\subseteq$": Let the bipatite graph $G = (X, Y, E)$, where X and Y are the sets of variables of R and S, respectively. We show that G is convex. A variable x_i associated with a tuple (a_i, c_i) of R is paired with a variable y_j associated with a tuple (b_j) of S if and only if $a_i < b_j < c_i$. Thus, if we enumerate the Y-variables ascendingly, the X-neighborhoods are sets of consecutive y_j's (that is, intervals).

"$\supseteq$": We are given a convex bipatite graph $G = (X, Y, E)$. We create a 2-DNF ϕ over the variable sets $[X, Y]$. W.l.o.g., let Y be the variable set for which an order exists such that the neighbourhoods $N(x_i)$ are sets of consecutive y_j's. Then, for each x_i there exist $1 \leq s_i \leq t_i \leq |Y|$ such that $N(x_i) = \{y_{s_i}, \ldots, y_{t_i}\}$. Now create a database D with relations $R(A, B)$ and $S(C)$, where X is the set of variables of R and Y is the set of variables of S, according to the following scheme: For each $x_i \in X$ add a corresponding tuple $(s_i - 0.1, t_i + 0.1)$ to R. For each $y_j \in Y$ add a corresponding tuple (j) to S. Then $x_i y_j \in \phi$ if and only if $y_j \in N(x_i)$. $\square$

The following lemma states that, as for multi-chain k-DNFs, the lineage of the single-guard query, which captures convex bipartite graphs, can be compiled into BDDs in polynomial time. The proof of this lemma is deferred to the master's thesis of the first author [30].

Lemma 5.9. *The lineage ϕ of the single-guard query on any tuple-independent probabilistic database can be compiled into a BDD in $O(|\phi|^2)$.*

Further results not discussed here include the tractability of the two-edge query $Q : -R(A, B), S(C, D), A < C, B < D$, which is not an one-out query. Besides the tractability of this query, it is shown in [30] that its lineage family strictly includes that of the single-guard query, and that its class of lineage graphs is strictly included in the class of weakly chordal bipartite graphs.

6. CONCLUSION AND FUTURE WORK

This paper shows the tractability of the new class of acyclic one-out inequality queries on tuple-independent probabilistic databases. This result strictly subsumes an earlier tractability result for inequality queries [20]. By exploiting the connection between structural properties of bipartite chain and convex graphs and the lineage of tractable queries, we are able to obtain novel tractability results for counting vertex covers in such graphs. The technique underlying our results is based on a syntactical characterization of k-DNF formulas that capture the lineage of tractable queries and that can be compiled in at most quadratic time into binary decision diagrams of size linear in the size of the formulas.

We plan to further investigate the tractability of queries beyond acyclic one-out. Two results concern the tractable single-guard and two-edge queries discussed in Section 5. It is not yet clear how to integrate these two query patterns with the acyclic one-out queries. In earlier work we have shown that inequality queries with two single-guard queries can easily become #P-hard [20].

7. 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] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. Mystiq: A system for finding more answers by using probabilities. In *SIGMOD Conference*, 2005.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[5] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[6] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

[7] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, 2007.

[8] A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4), 2001.

[9] A. Darwiche and P. Marquis. "A knowlege compilation map". *Journal of AI Research*, 17:229–264, 2002.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] J. Huang, L. Antova, C. Koch, and D. Olteanu. Maybms: A probabilistic database management system. In *Proc. SIGMOD*, 2009.

[12] J. Huang and A. Darwiche. Using DPLL for efficient OBDD construction. In *Revised Selected Papers of SAT 2004*, 2005.

[13] N. Ishakbeyoglu and Z. Ozsoyoglu. Testing satisfiability of a conjunction of inequalities. In *Int. Symposium on Computer and Inf. Syst. (ISCIS XII)*, pages 148–154, 1997.

[14] R. Jampani, M. W. F. Xu, L. L. Perez, C. M. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proc. SIGMOD*, pages 687–700, 2008.

[15] R. M. Karp, M. Luby, and N. Madras. "Monte-Carlo Approximation Algorithms for Enumeration Problems". *J. Algorithms*, **10**(3):429–448, 1989.

[16] M.-S. Lin. Fast and simple algorithms to count the number of vertex covers in an interval graph. *Information Processing Letters*, 102:143–146, 2007.

[17] C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag, 1998.

[18] Y. Okamoto, T. Uno, and R. Uehara. Linear-time counting algorithms for independent sets in chordal graphs. In *Proc. of 31st Int. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 433–444, 2005.

[19] D. Olteanu and J. Huang. Using obdds for efficient query evaluation on probabilistic databases. In *Proc. of 2nd Int. Conf. on Scalable Uncertainty Management (SUM),*, Oct 2008.

[20] D. Olteanu and J. Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *ACM SIGMOD 2009*, 2009.

[21] D. Olteanu, J. Huang, and C. Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *Proc. ICDE*, pages 640–651, 2009.

[22] S. J. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[23] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. ICDE*, 2007.

[24] C. Ré and D. Suciu. The trichotomy of having queries on a probabilistic database. *Accepted to VLDB Journal*, 2009.

[25] U. Rostock. Information system on graph class inclusions. http://wwwteo.informatik.uni-rostock.de/isgci/.

[26] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proc. ICDE*, 2007.

[27] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: Native support for uncertain data (demo). In *Proc. SIGMOD*, 2008.

[28] M. Soliman, I. Ilyas, and K. Chang. Probabilistic top-k and ranking-aggregate queries. *ACM TODS*, 33, 2008.

[29] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. In *Proc. VLDB*, 2008.

[30] R. Wissmann. Tractable queries with inequalities on probabilistic databases. Master's thesis, Oxford University Computing Laboratory, 2009. available at http://web.comlab.ox.ac.uk/projects/SPROUT/.