On the Optimal Approximation of Queries Using Tractable Propositional Languages

Robert Fink and Dan Olteanu
Oxford University Computing Laboratory
{robert.fink,dan.olteanu}@comlab.ox.ac.uk

ABSTRACT

This paper investigates the problem of approximating conjunctive queries without self-joins on probabilistic databases by lower and upper bounds that can be computed more efficiently. We study this problem via an indirection: Given a propositional formula Φ, find formulas in a more restricted language that are greatest lower bound and least upper bound, respectively, of Φ. We study bounds in the languages of read-once formulas, where every variable occurs at most once, and of read-once formulas in disjunctive normal form.

We show equivalences of syntactic and model-theoretic characterisations of optimal bounds for unate formulas, and present algorithms that can enumerate them with polynomial delay. Such bounds can be computed by queries expressed using first-order queries extended with transitive closure and a special choice construct.

Besides probabilistic databases, these results can also benefit the problem of approximate query evaluation in relational databases, since the bounds expressed by queries can be computed in polynomial combined complexity.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*

General Terms

Algorithms, Languages, Theory

Keywords

Probabilistic database, Query approximation

1. INTRODUCTION

Approximate query evaluation is preferred over exact query evaluation in cases when the exact evaluation is too expensive and quick approximate answers are favoured over delayed exact answers. This paper investigates the problem of query approximation through the glasses of propositional formula approximation. Formulas and queries are intimately connected in the following way: Given an input database where every tuple is annotated by a distinct variable, each tuple t in the query answer is annotated by a formula over the input tuples that contributed to t.

We seek bounds expressible in restricted classes of propositional formulas that are model-based, optimal, efficiently computable, and allow for efficient probability computation, in case they are defined over random variables. Bounds are defined semantically based on the set of models (or satisfying assignments) of formulas. The set of models of a formula is included in the set of models of any of its upper bounds and includes the set of models of any of its lower bounds. Lower and upper bounds are optimal with respect to a language if there are no other lower and upper bounds, respectively, in that language whose sets of models are closer to the set of models of the formula to approximate.

In this paper, bounds are expressed in the languages of read-once formulas or of its intersection with the language of formulas in disjunctive normal form (DNF). Read-once formulas are propositional formulas where every variable occurs at most once, that is, they are in *1-Occurrence Form*. We denote this class by 1OF. The DNF restriction of 1OF is denoted here by *independent DNF*, or iDNF for short.

The proposed approximation framework is relevant to database scenarios that can benefit from optimal and efficient approximation of query evaluation. We discuss three such scenarios: approximate probability computation in probabilistic databases, approximation of provenance information and explanations in provenance databases, and approximate query evaluation in relational databases.

Scenario 1: Probabilistic Databases

Our main motivation for this work comes from probabilistic databases. The problem of query evaluation in probabilistic databases has received tremendous attention in recent years, a fairly recent survey was compiled by Dalvi, Ré, and Suciu [2]. This problem is known to be #P-hard already for simple conjunctive queries and restricted probabilistic databases. One particularly promising evaluation strategy for hard queries is to consider approximate evaluation.

Consider so-called tuple-independent probabilistic databases, where each tuple is associated with a distinct Boolean random variable. By query evaluation techniques reminiscent of conditioned tables [14], propositional formulas called *lineage* can be computed along with the query answers and may subsequently be used to compute answer probabilities. Figure 1 shows the lineage of three queries evaluated on a probabilistic database. The lineages of positive queries are

R	A
x_1	1
x_2	2
x_3	3

S	A	B
y_1	1	1
y_2	1	2
y_3	2	1
y_4	2	2
y_5	3	3
y_6	3	4

T	B
z_1	1
z_2	2
z_3	3
z_4	4

Query Q_1:-$R(A), S(A, B), T(B)$ has lineage

$$\Phi_1 = x_1y_1z_1 \lor x_1y_2z_2 \lor x_2y_3z_1 \lor x_2y_4z_2 \lor x_3y_5z_3 \lor x_3y_6z_4.$$

Query Q_2:-$R(A), S(A', B), T(B), A \leq A'$ has lineage

$$\Phi_2 = x_1y_1z_1 \lor x_1y_2z_2 \lor x_1y_3z_1 \lor x_1y_4z_2 \lor x_1y_5z_3 \lor x_1y_6z_4 \lor \\ x_2y_3z_1 \lor x_2y_4z_2 \lor x_2y_5z_3 \lor x_2y_6z_4 \lor x_3y_5z_3 \lor x_3y_6z_4.$$

Query Q_3:-$R(A), S(A, B)$ has lineage

$$\Phi_3 = x_1y_1 \lor x_1y_2 \lor x_2y_3 \lor x_2y_4 \lor x_3y_5 \lor x_3y_6 \\ = x_1(y_1 \lor y_2) \lor x_2(y_3 \lor y_4) \lor x_3(y_5 \lor y_6).$$

Figure 1: Relations R, S, T annotated with variables from disjoint sets $V_1 = \{x_1, \ldots, x_3\}$, $V_2 = \{y_1, \ldots, y_6\}$, $V_3 = \{z_1, \ldots, z_4\}$, respectively. Lineages Φ_1 to Φ_3 of four queries Q_1 to Q_3 evaluated on this database.

unate DNF formulas. A key contribution of the this work is an approach that can efficiently derive optimal lower bounds Φ_L and upper bounds Φ_U in tractable languages such as iDNF or 1OF for any unate DNF formula Φ. Moreover, we can lift this result to the query level by showing how lineage bounds can be computed by queries.

Let $\mathcal{M}$ and P be functions that map formulas to their sets of models and to their probabilities, respectively. Model-based bounds imply probability-based bounds for any probability distribution over the Boolean random variables:

Proposition 1. *Let Φ, Φ_L, and Φ_U be formulas. Then,*

$$\mathcal{M}(\Phi_L) \subseteq \mathcal{M}(\Phi) \subseteq \mathcal{M}(\Phi_U) \Rightarrow P(\Phi_L) \leq P(\Phi) \leq P(\Phi_U).$$

We devise algorithms that can enumerate optimal bounds efficiently. Such bounds cannot lead, however, to approximations with arbitrary precision, since exact probability computation is #P-hard already for unate bipartite DNF formulas. Efficiently computable bounds can be nevertheless very useful in conjunction with known procedures that incrementally decompose the input formula Φ into subformulas and compute bounds on $P(\Phi)$ based on bounds of these subformulas [19, 7]. This is repeated until the desired precision is reached. Efficient computation of good probability bounds for intermediate decomposition steps is essential to the effectiveness of such techniques.

The closest in spirit to our approach is a technique developed independently by Gatterbauer, Abhay, and Suciu [9]. Their technique computes upper bounds for probabilities of conjunctive queries without self-joins. These bounds are not model-based, and in particular not optimal. Although the query lineage is used to compute the upper bounds, the interpretation of this lineage is non-standard: Each literal is intepreted as being a unique variable.

Scenario 2: Provenance Databases

Our work is applicable to the approximation of provenance information in the answers to queries on databases with annotations. Similarly to the probabilistic case, query lineage encodes symbolically all possible explanations for the existence of a tuple in the query answer in terms of combinations of the input tuples. Lineage bounds can then represent coarser, more compact, and efficiently computable explanations of the query answer. Lower bounds are *correct but possibly incomplete* explanations, while upper bounds are *complete but potentially incorrect* explanations in the following sense: Every explanation for a lower bound is necessarily an explanation for the answer; however, there may exist explanations for the answer that are not explanations for the lower bound. The situation for upper bounds is symmetric.

Scenario 3: Relational Databases

Existing work on approximate query answering in relational databases considers the use of synopses (histograms, join synopses, and wavelets) to speed-up the evaluation of aggregate queries with the goal of quickly reporting the leading digits of the answers. A survey of these synopsis-based techniques is given by Garofalakis and Gibbons [8]. Their focus is not on deriving optimal bounds within a given language.

Given a query Q in a language $\mathcal{QL}$, the approximation problem we consider in the relational setting is to compute two queries Q_L and Q_U in a language $\mathcal{QL}'$ such that $Q_L(D) \subseteq Q(D) \subseteq Q_U(D)$ for any relational database D and the computational complexity for $\mathcal{QL}'$ is lower than for $\mathcal{QL}$.

When applied to relational databases, our approximation approach comes with the following twist. We can derive queries Q_L and Q_U such that their answers can be represented compactly as a 1OF or iDNF expression over tuples. Such answer representations can be computed in polynomial combined complexity and allow for the enumeration of (possibly exponentially many) answers with polynomial delay.

Contributions

The main contributions of this paper are as follows:

- We introduce a query approximation framework based on novel results on model-based optimal approximation of unate formulas. We investigate two languages to express lower and upper bounds: read-once formulas and their DNF retrictions.
- We give syntactic characterisations of optimal lower and upper bounds in iDNF and 1OF, as well as algorithms that can enumerate such bounds for unate DNF formulas with polynomial delay.
- We show that the computation of model-based bounds on the lineage of conjunctive queries without self-joins is expressible using queries that can be evaluated with polynomial combined complexity using iDNF or 1OF representations of the query answers.

Due to lack of space, the proofs of the statements made in this paper are given in an extended technical report [6].

2. PRELIMINARIES

We use the language of propositional formulas over a finite set of Boolean variables with its standard interpretation.

Syntax of Formulas. We assume the standard syntactic notions of variables, constants 1 (*true*) and 0 (*false*), literals, and (propositional) formulas constructed by combining literals or constants using the logical connectives $\lor$ and $\land$. In this paper, we consider unate formulas, i.e. formulas in which all literals are positive. A clause is a conjunction of literals. A formula in disjunctive normal form (DNF) is a disjunction of clauses. At the syntactic level, we see clauses as sets of literals and DNF formulas as sets of clauses. This explains set notations such as $\subseteq$ on clauses and formulas. Consequently, $\Phi \subseteq \Psi$ means that Φ consists of a subset of the clauses or literals of Ψ, if Φ and Ψ are DNF formulas or clauses, respectively. The constants 0 and 1 are special sets such that 1 is contained in any set and 0 contains any set. Two clauses are (syntactically) *independent* if their variable sets are disjoint. We use $vars(\Phi)$ to denote the set of variables of a formula Φ.

Definition 1. Let $(V_1, \dots, V_k)$ be a tuple of disjoint sets of variables. A formula Φ is *k-partite* over $(V_1, \dots, V_k)$ if each clause in Φ has the form $v_1 \wedge \dots \wedge v_k$, where $v_i \in V_i$ for all $1 \leq i \leq k$. The *projection* $\Phi_{i,j}$ of Φ is a formula obtained from Φ by removing all literals not in V_i or V_j. $\square$

Example 1. Figure 1 shows 3-partite unate DNF formulas Φ_1 and Φ_2 over the tuple of disjoint variable sets (V_1, V_2, V_3). Formula Φ_3 is a 2-partite over (V_1, V_2). $\square$

The graph of a formula Φ, denoted by $G(\Phi)$, has one node for each distinct variable in Φ, and one edge (x, y) for each pair of literals x, y that appear together in a clause of Φ. For k-partite formulas, we use a more specific representation.

Definition 2. Let Φ be a k-partite formula over $V_1, \dots, V_k$. The bipartite graph $B_{i,j} = (V_i, V_j, E_{ij})$ of projection $\Phi_{i,j}$ of Φ has an edge (x, y) for each clause $xy \in \Phi_{i,j}$. The *set of projection graphs* of Φ is $\mathcal{B}(\Phi) = \bigcup\{B_{i,j} \mid 1 \leq i < j \leq k\}$. $\square$

For a given k-partite formula with n variables and m clauses, its graph has n nodes and $O(m \cdot k^2)$ edges. Figure 2 depicts the graphs of the formulas from Figure 1.

Semantics of Formulas. We assume the standard semantic notion of satisfying assignments, also called models, of a formula. Let $\mathcal{M}$ be a function mapping formulas to their sets of models. Given two formulas Φ and Ψ over the same variables, Φ is *semantically* contained in Ψ, denoted by $\Phi \models \Psi$, if $\mathcal{M}(\Phi) \subseteq \mathcal{M}(\Psi)$. We also say that Φ *implies* Ψ. Equivalence is two-way containment.

Linking Semantics and Syntax. Given two clauses φ and ψ, it holds that $\varphi \models \psi$ if and only if $\varphi \supseteq \psi$. For DNF formulas, we will use the following key result:

LEMMA 1. [22] Let Φ and Ψ be two DNF formulas. Then

$$\Phi \models \Psi \quad \text{iff} \quad \forall \varphi \in \Phi\ \exists \psi \in \Psi \ : \ \varphi \models \psi. \square$$

A unate DNF formula Φ is *reducible* if there exist clauses $\varphi, \psi \in \Phi$ that satisfy $\varphi \models \psi$, and *irreducible* otherwise. In the former case, φ is redundant in Φ and can be removed.

Queries on Annotated and Probabilistic Databases. We consider conjunctive queries without self-joins expressed using Datalog syntax. By default, equi-joins are the only type of joins allowed in such queries. Figure 1 shows three Boolean queries. Query Q_1 has two query variables A and B that express joins between R, S, and T. Query Q_2 uses an inequality join $A \leq A'$.

An annotated database is a relational database, where each tuple is annotated by a propositional formula, sometimes called *lineage*. If all formulas in an annotated database are pairwise independent, then the database is called *tuple-independent*. Such annotated databases have been extensively studied in the context of provenance in databases, incomplete databases [14], probabilistic databases, and in a unifying framework by Green et al. [12].

The evaluation of queries on annotated databases follows standard query evaluation in relational databases, with the addition that the annotations of the input tuples are propagated to the answer tuples they contributed to. The annotation of a distinct tuple created (a) by a join of two input tuples is the conjunction of the annotations of these input tuples, and (b) by a projection or union is the disjunction of the annotations of the input tuples. By construction, the lineage of a conjunctive query wihtout self-joins and with k relation symbols is thus a unate k-partite DNF formula defined over variable sets $(V_1, \dots, V_k)$, with one variable set V_i per relation symbol.

In probabilistic databases, the lineage is defined over random variables and can be used to compute the probabilities of tuples in the answers to queries: Given a tuple t in the answer to query Q with lineage Φ, the probability of t is equal to the probability $P(\Phi)$ that a random valuation of the variables in Φ satisfies Φ. Computing $P(\Phi)$ is a #P-hard problem for conjunctive queries, though there are known classes of queries for which the problem is in polynomial time, e.g. [3]. We next recall such a tractable class.

For a query variable A in a query Q, we denote by $s(A)$ the set of relation symbols in Q that mention A.

Definition 3. A conjunctive query Q is *hierarchical* if for any two non-head query variables A and B, either $s(A) \cap s(B) = \emptyset$, or $s(A) \subseteq s(B)$, or $s(A) \supset s(B)$. $\square$

In the class of conjunctive queries without self-joins, the hierarchical queries are the only tractable queries [3].

Example 2. Consider the queries from Figure 1. Query Q_1 is not hierarchical since $s(A) = \{R, S\}$, $s(B) = \{S, T\}$. Query Q_3 is hierarchical, since $s(B) \subset s(A)$ and A and B are the only query variables. Query Q_2 has an inequality join; there are extensions of the hierarchical property to cope with inequality joins [18]. $\square$

Assumptions and conventions. Unless explicitly stated otherwise, *we assume formulas to be unate and irreducible* by default. For arbitrary formulas Φ, set-based notions like $\varphi \in \Phi$ or $x \in \varphi$ are thus with respect to the unique irreducible model-equivalent DNF formula of Φ. If a unate formula is given in DNF, computing the irreducible equivalent DNF can be done in polynomial time. Turning formulas into DNF representations has exponential overhead in worst-case.

Lowercase Latin letters denote variable symbols, capital Latin letters denote query variables, and uppercase (lowercase) Greek letters denote formulas (clauses).

3. FORMULA APPROXIMATION

The core problem investigated in this paper is: Given a formula Φ from a propositional language, find formulas from a more restrictive language that approximate Φ.

Definition 4. Let $\mathcal{L}'$ and $\mathcal{L}$ be languages of propositional formulas such that $\mathcal{L}' \subset \mathcal{L}$ and $\Phi \in \mathcal{L}$. Formulas $\Phi_L, \Phi_U \in \mathcal{L}'$ are *lower and upper bounds for Φ with respect to $\mathcal{L}'$*, if $\mathcal{M}(\Phi_L) \subseteq \mathcal{M}(\Phi)$ and $\mathcal{M}(\Phi) \subseteq \mathcal{M}(\Phi_U)$ respectively.

If in addition there are no formulas $\Phi'_L, \Phi'_U \in \mathcal{L}'$ such that

$$\mathcal{M}(\Phi_L) \subset \mathcal{M}(\Phi'_L) \subseteq \mathcal{M}(\Phi) \text{ and } \mathcal{M}(\Phi) \subseteq \mathcal{M}(\Phi'_U) \subset \mathcal{M}(\Phi_U),$$

then Φ_L is a *greatest lower bound (GLB)* and Φ_U is a *least upper bound (LUB) for Φ with respect to $\mathcal{L}'$*. $\square$

The notions GLB and LUB provide an intuitive semantic characterisation of *optimal* bounds. We consider languages $\mathcal{L}'$ such that (i) we can efficiently find optimal bounds in $\mathcal{L}'$, and (ii) $\mathcal{L}'$ allows for efficient key computational tasks such as probability computation. The first language $\mathcal{L}'$, called *one-occurrence form* (1OF), is that of formulas in which every variable occurs at most once, and of formulas that are equivalent to formulas in which every variable occurs at most once. The second language $\mathcal{L}'$, called *independent DNF* (iDNF), is the intersection of 1OF and DNF.

Example 3. The formula $x_1y_1 \lor x_1y_2$ is in DNF, but not in iDNF, since the two clauses have the variable x_1 in common. This formula and its equivalent $x_1(y_1 \lor y_2)$ are in 1OF. Formula Φ_3 from Figure 1 is in 1OF. □

The 1OF language has a long history and many names, such as read-once functions [10], fanout-free functions, or non-repeating trees [21], and also many applications including logic synthesis and circuit design [21] and probabilistic databases [17]. Problems such as satisfiability, model counting, and probability computation are hard for general propositional formulas are tractable for 1OFs. The 10F property is decidable in polynomial time for DNF formulas. Furthermore, the equivalent (syntactic) 1OF can be found in polynomial time and is unique up to commutativity of the binary connectives [21]. The 1OF language is particularly relevant in the context of probabilistic databases, since the lineage of any tractable relational algebra query without repeating symbols on tuple-independent probabilistic databases is in 1OF and its probability can be computed using relational query plans [17, 7].

Languages beyond 1OF lose tractability. It is NP-hard to decide if a given unate formula admits an equivalent formula where every variable occurs at most twice (read-twice formulas) [5]; this is still open in case of input DNF formulas. In case of formulas where every variable occurs at most four times, model counting (and hence probability computation) is already #P-hard [26]. A different direction is to consider complete languages, i.e. languages that can represent any propositional formula either directly or in a formalism that preserves its models, and that still allow for efficient probability computation. A prime example of such languages is the family of Binary Decision Diagrams, including OBDDs, FBDDs, and d-DNNFs [4]. Such languages allow for several equivalent representations of the same input formula but with an exponential gap between their sizes. Finding a minimal representation for an input formula is NP-hard [16].

Remark 1. There is a solid body of work on approximating various computational tasks on propositional formulas. Exact and approximate approaches to model counting, which are also relevant to probabilistic databases, are surveyed by Gomes et al. [11]. These approaches however do not consider the approximation of propositional formulas by formulas in a more restrictive language, as done here. The closest in spirit to our work is that by Selman on approximating CNF theories by model-based lower and upper bound conjunctions of Horn clauses [23]. The languages iDNF and 1OF are incomparable to conjunctions of Horn clauses. □

Before we discuss optimal iDNF and 1OF approximations, we give a syntactic characterisation of (not necessarily optimal) lower and upper bound formulas for arbitrary unate formulas: *All* bounds for a given unate formula Φ can be defined by simple *syntactic* manipulations of Φ.

PROPOSITION 2. *Let Φ, Φ_L, and Φ_U be unate formulas.*

- *Φ_L is a lower bound of Φ if and only if Φ_L can be obtained by removing clauses from Φ or by adding literals to its clauses. Removing all clauses results in the lower bound 0.*
- *Φ_U is an upper bound of Φ if and only if Φ_U can be obtained by adding clauses to Φ or by removing literals from its clauses. Removing all literals from a clause results in the upper bound 1.* □

These syntactic manipulations necessarily lead to lower and upper bounds. Indeed, by removing clauses from a formula we reduce its set of models. By adding variables to clauses, we further constrain their satisfiability and hence reduce the set of models. In both cases, one obtains formulas that are lower bounds. The inverse manipulations lead to upper bounds. More importantly, Proposition 2 states that all bounds can be gained in this way, a fact that follows from Lemma 1. Section 4 shows that it is not necessary to add literals to clauses in order to find *optimal* iDNF lower bounds, while removing variables can be required in order to get to optimal iDNF upper bounds. Section 5 shows how to obtain optimal 1OF bounds that can be constructed by solely removing or adding clauses.

Example 4. Neither formula Φ_1 nor Φ_2 from Figure 1 are in iDNF. The following iDNF formulas are lower and upper bounds for Φ_1 (several others are possible):

$$\Phi_L = x_1y_1z_1 \lor x_2y_4z_2 \lor x_3y_5z_3,\ \Phi_U = z_1 \lor x_1y_2 \lor x_2y_4z_2 \lor x_3.$$

Indeed, Φ_L is a lower bound since it is a subset of Φ. Each clause in Φ implies at least one clause of Φ_U, hence Φ_U is an upper bound. A closer inspection reveals that each clause in Φ_U can be obtained by dropping literals from clauses in Φ. As we will show in Section 4, the bounds Φ_L and Φ_U are already optimal for iDNF. They are, however, not optimal for 1OF, since they can be improved as follows:

$$\Phi'_L = x_1y_1z_1 \lor x_2y_4z_2 \lor x_3(y_5z_3 \lor y_6z_4).$$
$$\Phi'_U = (y_1 \lor y_3)z_1 \lor (x_1y_2 \lor x_2y_4)z_2 \lor x_3(y_5z_3 \lor y_6z_4).$$ □

4. OPTIMAL IDNF APPROXIMATIONS

We characterise syntactically the optimal iDNF bounds for unate formulas, and give algorithms that find such bounds.

4.1 iDNF Greatest Lower Bounds

Our main tool used to find optimal iDNF lower bounds is the syntactic notion of maximal lower bounds.

Definition 5. Let Φ be a unate formula. An iDNF formula Φ_L is called a *maximal lower bound (MLB)* for Φ if it satisfies the following conditions:

1. (*Lower bound*) Φ_L contains a subset of the clauses of Φ: $\Phi_L \subseteq \Phi$;
2. (*Maximality*) Φ_L cannot be further extended: There is no clause $\varphi \in \Phi$ such that $vars(\varphi) \cap vars(\Phi_L) = \emptyset$. □

The first criterion ensures that Φ_L is indeed an iDNF lower bound. The maximality property enforces that it is a *maximal* lower bound. An MLB for a formula Φ is thus a maximal set of pairwise independent clauses of Φ.

Example 5. The MLBs for formula Φ_1 from Figure 1 are:

$$x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_5z_3, \quad x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_6z_4$$
$$x_1y_2z_2 \vee x_2y_3z_1 \vee x_3y_5z_3, \quad x_1y_2z_2 \vee x_2y_3z_1 \vee x_3y_6z_4 \qquad \square$$

With respect to the iDNF language, the maximal lower bounds correspond precisely to the greatest lower bounds. This is particularly important, since MLB is a syntactic notion defined in terms of sets of clauses, and GLB is a semantic notion defined in terms of sets of models.

THEOREM 1. *Let Φ be a unate formula. An iDNF formula Φ_L is a maximal lower bound for Φ if and only if Φ_L is a greatest lower bound for Φ.*

The implication GLB $\Rightarrow$ MLB holds since if any of the two MLB conditions fail, then we cannot obtain a GLB. For the other direction, it can be shown that no strict upper bound Φ'_L for Φ_L is a lower bound for Φ. The latter case has two sub-cases: There is a clause $\varphi \in \Phi$ that is in Φ'_L and either is or is not in Φ_L. In the first sub-case, φ necessarily shares variables with one clause in $\varphi_L \in \Phi_L$, and, according to Lemma 1, it must be contained in φ_L. But then, φ must be φ_L, otherwise Φ_L can be further extended and hence is no MLB. The second sub-case follows similarly.

The syntactic definition of MLBs suggests an algorithm for enumerating all GLBs by recursively constructing all maximal subsets of non-conflicting clauses of Φ. This algorithm may need time exponential in the number of clauses of Φ. The following proposition shows that this is necessarily so:

PROPOSITION 3. *The unate DNF formula*

$$\Phi = (x_1y_1 \vee x_1y_2) \vee \cdots \vee (x_ny_{2n-1} \vee x_ny_{2n})$$

has $2n$ clauses and 2^n iDNF GLBs. Any formula containing either x_iy_{2i-1} or x_iy_{2i} for all $i \leq n$ is an iDNF GLB of Φ.$\square$

Since there can be exponentially many GLBs that are by definition incomparable with respect to their model sets, it may be desirable to find "the best" GLB for a formula according to different criteria. Possible rankings of GLBs are on the number of clauses or, an arguably more useful criterion, on their probabilities, provided they are defined over random variables. Additionally, it can be useful to have an enumeration of all GLBs with polynomial delay [15], which means that the time before finding the first GLB, as well as the time between every two consecutive GLBs, is polynomial only in the input size, and not in the number of GLBs. We obtain results for ranking and enumerating GLBs by exploiting the following correspondence between MLBs and independent sets in graphs.

Definition 6. Let Φ be a DNF formula. The *clause-dependency graph* of Φ is a graph $G = (\Phi, E)$, where nodes are clauses of Φ, and edges represent pairs of dependent clauses:

$$E = \{(\varphi, \psi) \mid \varphi, \psi \in \Phi \text{ and } vars(\varphi) \cap vars(\psi) \neq \emptyset\}.\square$$

LEMMA 2. *Let Φ be a unate DNF formula. A formula $\Phi_L \subseteq \Phi$ is an iDNF greatest lower bound for Φ if and only if the clause-dependency graph of Φ_L is a* maximal independent set *in the clause-dependency graph of Φ.*

A similar equivalence can be established between maximum weighted independent sets and iDNF greatest lower bounds with maximum probability. Therefore, known results for independent sets, such as enumeration of maximal independent sets with polynomial delay [25, 15], immediately carry over to the case of iDNF greatest lower bounds.

```
MUB (Unate formula $\Phi$, iDNF upper bound $\Phi_U$)
outputs all iDNF MUBs of $\Phi$

$\Phi_0 \leftarrow \{\varphi_u \in \Phi_U : \varphi_u$ has no critical witness in $\Phi\}$
mergeCandidates $\leftarrow \{(\varphi_u, w, \varphi'_u) \mid w \in \Phi; \varphi_u, \varphi'_u \in \Phi_0;$
                        $w$ is witness of $\varphi_u$ and $\varphi'_u\}$
if $\Phi_0 \neq \emptyset$ then
    // Remove clauses without crit. witness
    foreach $\varphi_u \in \Phi_0$ do MUB$(\Phi, \Phi_U \setminus \{\varphi_u\})$
    // Merge clauses without critical witness
    foreach $(\varphi_u, w, \varphi'_u) \in$ mergeCandidates do
        MUB$(\Phi, (\Phi_U \setminus \{\varphi_u, \varphi'_u\}) \cup \{\{\varphi_u \cup \varphi'_u\}\})$
else
    $V \leftarrow vars(\Phi) \setminus vars(\Phi_U)$
    Extend $\Phi_U$ by variables in $V$ in every possible way
    without altering the witness relations
    Output $\Phi_U$
```

Algorithm 1: Finding all MUBs of a unate formula Φ. Initial call MUB(Φ, $\bigcup_{v \in vars(\Phi)}\{\{v\}\}$).

COROLLARY 1 (LEMMA 2,[25, 15]). *The set of all iDNF greatest lower bounds for a unate DNF formula can be enumerated with polynomial delay.* $\square$

Since finding a maximum independent set is NP-hard, we cannot efficiently enumerate the maximal independent sets, or equivalently the iDNF GLBs, in decreasing order of their size or probabilities.

PROPOSITION 4. *Enumerating the iDNF greatest lower bounds of a unate DNF formula in decreasing order of the number of their clauses or their probabilities is NP-hard.* $\square$

Although we cannot efficiently obtain the iDNF GLB with maximum probability, a constant-factor approximation can be obtained for input unate k-partite DNF formulas.

PROPOSITION 5. *Let Φ be a k-partite unate DNF formula. There exists a polynomial time algorithm that constructs an iDNF greatest lower bound Φ_L for Φ such that $P(\Phi_L^{opt}) \leq k \cdot P(\Phi_L)$, where Φ_L^{opt} is the iDNF greatest lower bound for Φ with the highest probability amongst all of Φ's iDNF greatest lower bounds.* $\square$

The algorithm constructs Φ_L by iterating over Φ's clauses in decreasing order of their probabilities and greedily selecting a maximal set of pairwise independent clauses. Φ_L thus contains the first clause φ_1 in the order, then the next clause independent of φ_1, and so on. In the construction of an upper bound for $P(\Phi_L^{\text{opt}})$, we use the fact that the maximum number of pairwise independent clauses between two clauses of Φ_L in the descending order of clauses is smaller than $k+1$.

4.2 iDNF Least Upper Bounds

Similar to the case of iDNF greatest lower bounds, we give a syntactic characterisation of iDNF least upper bounds for unate DNF formulas. We introduce a few necessary notions.

Definition 7. Let Φ and Ψ be formulas. A clause $\varphi \in \Phi$ is a *witness* for a clause $\psi \in \Psi$ if $\varphi \models \psi$. We also say that ψ *has the witness* φ. If in addition there is no clause $\psi' \in \Psi$ with $\psi \neq \psi'$ and $\varphi \models \psi'$, then φ *is a critical witness* for ψ. We then also say that ψ *has the critical witness* φ. $\square$

Example 6. Consider formula Φ_1 from Figure 1 and the formula $\Psi = x_1y_1 \vee x_2 \vee z_2 \vee x_1y_6$. Clause $x_1y_1 \in \Psi$ has the critical witness $x_1y_1z_1 \in \Phi_1$, clause $x_2y_4z_2$ is a non-critical witness for x_2 and $z_2 \in \Psi$, and finally clause $x_1y_6 \in \Psi$ has no witness in Φ_1. □

Definition 8. Let Φ and Ψ be formulas. The bipartite *witness graph* $\Gamma_{\Phi,\Psi} = (V, V', E)$ from Φ to Ψ is defined as follows: V is the set of clauses of Φ, V' is the set of clauses of Ψ, and there is a directed edge $(\varphi, \psi) \in E$ from a clause $\varphi \in \Phi$ to a clause $\psi \in \Psi$ if $\varphi \models \psi$. □

A witness graph $\Gamma_{\Phi,\Psi}$ shows which clauses of Φ imply which clauses of Ψ. The set of clauses with incoming edges thus represent an upper bound for the set of clauses with outgoing edges. A node $\varphi \in V$ is then a witness for all nodes $\psi \in V'$ with $(\varphi, \psi) \in E$, and it is a critical witness for a node $\psi \in V'$ if $(\varphi, \psi) \in E$ and there is no other node in V that is a witness for ψ. The following figure shows the connected nodes in the witness graph for formulas Φ_1, Ψ from Example 6.

$$\begin{tikzcd}
x_1y_1z_1 \arrow[r, "\text{critical}"] & x_1y_1 \\
x_2y_4z_2 \arrow[r] \arrow[rd] & x_2 \\
 & z_2
\end{tikzcd}$$

We are now ready to present our main syntactic tool for finding upper bounds.

Definition 9. Let Φ be a unate formula. An iDNF formula Φ_U is a *minimal upper bound (MUB)* for Φ if it satisfies the following conditions:

1. (*Upper bound*) Every $\varphi \in \Phi$ is a witness for some clauses in Φ_U;
2. (*Maximality*) There is no clause $\varphi_u \in \Phi_U$ that can be extended by a variable from $vars(\Phi)$ while preserving condition (1) and keeping Φ_U in iDNF;
3. (*Irreducibility*) Every clause $\varphi_u \in \Phi_U$ has some critical witnesses in Φ. □

Following Lemma 1, the first condition ensures that $\Phi \models \Phi_U$. The second condition ensures that we cannot obtain an iDNF refinement of Φ_U which is still an upper bound for Φ by making clauses in Φ_U more specific (thus narrowing the set of models of Φ_U). The third condition states that all clauses in Φ_U are necessary, and dropping any of them would lead to clauses in Φ violating Lemma 1.

This syntactic characterisation of upper bounds precisely matches the semantic notion of least upper bounds:

Theorem 2. *Let Φ be a unate formula. An iDNF formula Φ_U is a minimal upper bound for Φ if and only if Φ_U is a least upper bound for Φ.* □

The implication LUB $\Rightarrow$ MUB holds since, if any of the three MUB conditions fail, we cannot obtain an LUB. The other direction uses the fact that any difference between an MUB Φ_U for Φ and a potentially better upper bound Φ'_U consists of having a clause $\varphi'_u \in \Phi'_U$ strictly containing (syntactically) a clause $\varphi \in \Phi_U$. A case analysis on a variable $x \in \varphi'_u$ and $x \notin \varphi_u$ shows that Φ_U does not satisfy the syntactic characterisation of an MUB: Either the irreducibility condition fails, or a clause in Φ_U may be expanded by x to obtain a smaller bound.

```
PolyMUB (Unate formula $\Phi$, iDNF upper bound $\Phi_U$)
outputs iDNF MUBs of $\Phi$

$\Phi_0 \leftarrow \{\varphi_u \in \Phi_U : \varphi_u$ has no critical witness in $\Phi\}$
mergeCandidates $\leftarrow \{(\varphi_u, w, \varphi'_u) \mid w \in \Phi; \varphi_u, \varphi'_u \in \Phi_0;$
                                        $w$ is witness of $\varphi_u$ and $\varphi'_u\}$
if $\Phi_0 \neq \emptyset$ then
    foreach $(\varphi_u, w, \varphi'_u) \in$ mergeCandidates do
        foreach $v \in \varphi_u$ do
            foreach $v' \in \varphi'_u$ do $A_{v,v'} \leftarrow 1; A_{v',v} \leftarrow 1$
        if $\forall v \in vars(\varphi_u), \forall v' \in vars(\varphi'_u) : A_{v,v'} = 0$
        then MUB$(\Phi, (\Phi_U \setminus \{\varphi_u, \varphi'_u\}) \cup \{\{\varphi_u \cup \varphi'_u\}\})$
else
    Output $\Phi_U$
```

Algorithm 2: Enumerating a subset of all MUBs of a formula Φ with polynomial delay. Initial call PolyMUB(Φ, $\bigcup_{v \in vars(\Phi)}\{\{v\}\}$). The algorithm assumes the existence of a *global* variable A such that for each pair of variables $v, v' \in vars(\Phi)$, $A_{v,v'}$ is a boolean flag. Initially, $A_{v,v'} = 0$ for all v, v'.

Algorithm 1 constructs MUBs for a formula Φ by starting with a pessimistic upper bound $\Phi_U = \bigcup_{v \in vars(\Phi)}\{\{v\}\}$ and refining it recursively: In every recursion step, witness graphs are constructed for the current upper bound Φ_U and each tighter bound representing a refinement of it obtained by merging or removing clauses in Φ_U. The set *mergeCandidates* contains pairs of clauses of Φ_U together with their common witness that may be merged without violating the *upper bound* condition. Every bound produced by the algorithm is indeed an MUB: The *upper bound* criterion is trivially satisfied by the initial pessimistic bound and remains satisfied, because only non-critical clauses are merged or removed. The base case of the recursion is reached whenever all clauses in Φ_U have at least one critical witness (*irreducibility*), i.e., $\Phi_0 = \emptyset$ in the algorithm. Every bound is then expanded to satisfy the *maximality* property.

Example 7. The formula $\Phi^1_{1,u} = x_1y_1 \vee y_2 \vee x_2y_3z_1 \vee y_4z_2 \vee x_3y_5z_3 \vee y_6z_4$ is an MUB for Φ_1 from Figure 1. Every clause in $\Phi^1_{1,u}$ has exactly one witness in Φ_1 which is also critical. No clause can be extended further, since $vars(\Phi^1_{1,u}) = vars(\Phi_1)$. The formula $\Phi^2_{1,u} = x_1 \vee x_2 \vee x_3y_5z_3 \vee y_6z_4$ is also an MUB; here clauses x_1 and x_2 both have two critical witnesses. It can be checked that $\Phi^2_{1,u}$ cannot be extended by any variable from $vars(\Phi_1)$. □

As in the case of greatest lower bounds, we can enumerate *all* iDNF least upper bounds of a given unate formula Φ:

Proposition 6. *Algorithm 1 computes all iDNF least upper bounds of a unate formula.* □

Soundness is sketched above. For completeness, consider an arbitrary MUB Φ_U and for illustration purposes, imagine Algorithm 1 to be non-deterministic, i.e. instead of iterating over possibilities, it chooses one randomly. It is straightforward to see that there exists a computation branch that produces Φ_U: First, remove all variables that are not in Φ_U, then merge clauses. These merging and removing steps necessarily have to occur intertwined. Consider formula $ax \vee bx \vee ay \vee by$. The LUB $a \vee b$ cannot be obtained if

```
1OF-LUB ($k$-partite unate formula $\Phi$)
outputs 1OF LUBs of $\Phi$ that are $k$-partite

$\mathcal{B} \leftarrow$ set of projections graphs.
Complete connected components of each projection
graph in $\mathcal{B}$.
$\mathcal{M} \leftarrow$ subset of all projection graphs $\mathcal{B}$ that contain a
component that is not aligned with components of a
different projection graph in $\mathcal{B}$.
if $\mathcal{M} = \emptyset$ then Output the 1OF represented by $\mathcal{B}$.
foreach projection graph $B \in \mathcal{M}$ do
    $\mathcal{B}' \leftarrow$ copy of $\mathcal{B}$.
    while there exist two projection graphs in $\mathcal{B}'$ that
    have non-aligned components $C, C'$ do
        // Wlog assume that the
        // projection graph of $C$ is not $B$.
        Align $C$ to $C'$ in $\mathcal{B}'$.
        Complete connected components of the
        projection graph of $C$ in $\mathcal{B}'$.
    Output the 1OF represented by $\mathcal{B}'$.

Complete projection graph $B$ in $\mathcal{B}'$
foreach Connected component $C$ in $B$ do
    Replace component $C$ with $(V, W, V \times W)$ in $\mathcal{B}'$.

Align component $C$ to component $C'$ in $\mathcal{B}'$
if $C$ and $C'$ are not aligned then
    A variable set $V$ of $C = (V, W, E)$ is incomparable
    with a variable set $V'$ of $C' = (V', W', E')$,
    i.e. $V \cap V' \neq \emptyset$ and $V \not\subseteq V'$ and $V \not\supseteq V'$.
    $D \leftarrow (V - V') \times W'$.
    Add edges $D$ to projection graph of $C'$ in $\mathcal{B}'$.
```

Algorithm 3: Finding 1OF LUBs of a k-partite unate formula Φ.

clauses are merged first, and the LUB $ax \vee b \vee y$ cannot be obtained if clauses are removed first.

The intertwined occurrence of merging, removing, and adding clauses makes the design of a polynomial-delay algorithm for enumerating MUBs difficult, since different computation branches may not necessarily lead to different MUBs.

Example 8. Consider the formula $ab \vee ax \vee xy$. Starting with the usual pessimistic bound $a \vee b \vee x \vee y$, the MUB $ab \vee x$ is found in a computation branch that first merges a and b and then removes y. A second branch might first remove a and y, and then expand clause b by a. A third branch will delete y and then merge a and b. All these branches lead to the same MUB $ab \vee x$. □

Polynomial delay can be obtained, however, if we only enumerate MUBs that have all input variables:

Proposition 7. *Given a unate DNF formula Φ, the set of all iDNF least upper bounds Φ_U for Φ such that $vars(\Phi) = vars(\Phi_U)$ can be enumerated with polynomial delay.* □

Algorithm 2 can be used to this effect. Soundness and completeness with respect to this restricted class of MUBs follow equivalently to Algorithm 1. The generated MUBs contain all variables of the input formula, since no variable ever gets dropped from the initial upper bound that contains all variables. It remains to be shown that the algorithm never returns two identical MUBs due to alternative merging orders that may lead to the same MUB and that the variables cannot be first deleted and later added again, as it can happen in Algorithm 1. The symmetric matrix A records the history of clause merges: $A_{v,v'} = 1$ if and only if an MUB has already been discovered in which variables v, v' appear in the same clause. The condition $A_{v,v'} = 0$ before merging clauses thus ensures that any new MUB does not have a clause with v and v'. Furthermore, it is guaranteed that every new MUB contains a clause in which v or v' are paired with some other variable. Since all MUBs are iDNF formulas, they are thus pairwise incomparable. Since all computation branches are bounded in depth by $|vars(\Phi)|$, return distinct MUBs, and are independent and may thus be removed from the stack after returning, and since matrix A has size $|vars(\Phi)|^2$, the algorithm needs space polynomial in the size of the input formula.

Example 9. The MUB $\Phi^2_{1,u}$ from Example 7 has fewer variables than Φ_1 and can thus not be found by Algorithm 2. Consider the iDNF formula $abc \vee xyz$ which is also its only MUB. The algorithm indeed finds this bound, for example by merging steps $ab \vee c \vee x \vee y \vee z$, $abc \vee x \vee y \vee z$, $abc \vee xy \vee z$, $abc \vee xyz$ and sets $A_{a,b} = A_{b,c} = A_{a,c} = A_{x,y} = A_{y,z} = A_{x,z} = 1$. Consequently, upon returning from the recursion to the top-level invocation of PolyMUB with $\Phi_U = a \vee b \vee c \vee x \vee y \vee z$, the for-loop will not invoke another recursive PolyMUB call, e.g., for $a \vee bc \vee x \vee y \vee z$, since $A_{b,c} = 1$. There is thus exactly one computation branch for one MUB. □

The time complexity of Algorithms 1 and 2 can be exponential in the size of the input, and this is necessarily so.

Proposition 8. *The unate DNF formula*

$$\Phi = (x_1 y_1 \vee x_1 y_2) \vee \cdots \vee (x_n y_{2n-1} \vee x_n y_{2n})$$

has $2n$ clauses and 3^n iDNF least upper bounds. Any formula consisting of either x_i, or $x_i y_{2i-1} \vee y_{2i}$, or $y_{2i-1} \vee x_i y_{2i}$ for all $1 \leq i \leq n$ is an iDNF least upper bound of Φ. □

5. OPTIMAL 1OF APPROXIMATIONS

In this section we show how to derive optimal 1OF bounds for unate k-partite formulas representing lineage of conjunctive queries without self-joins.

Recognizing 1OF formulas can be done using the following characterisation [13]: A unate formula Φ is in 1OF if and only if Φ is *normal* and $G(\Phi)$ is P_4-free. A graph is *P_4-free* if it has no induced subgraph that is isomorphic to P_4, i.e. the graph $\bullet - \bullet - \bullet - \bullet$. A formula is normal if it can be unambiguously recovered from its graph. For instance, in the graphs of Figure 2 (with edges (x_i, z_j) added), $G(\Phi_2)$ has the P_4 subgraph $x_1 - z_1 - x_2 - z_3$, and $G(\Phi_1)$ has the P_4 subgraph $x_1 - z_1 - x_2 - y_4$. The formulas Φ_1 and Φ_2 are thus not in 1OF. For unate DNF formulas, P_4-freeness and subsequent normality checks can be performed in polynomial time in their size [10]. The graph of a k-partite formula is always normal, if it is P_4-free [24].

A 1OF formula can be recovered from its graph G in polynomial time [13, 10]. We next sketch the implementation of a function ρ that maps graphs to 1OF formulas. If G is a set of connected components $G_1, \ldots, G_n$, then $\rho(G) = (\rho(G_1) \vee \cdots \vee \rho(G_n))$. Otherwise, we compute the complement of G and check again for connected components $G_1, \ldots, G_m$. Then, $\rho(G) = (\rho(G_1) \wedge \cdots \wedge \rho(G_m))$. We recurse with smaller graphs and check for connected components in

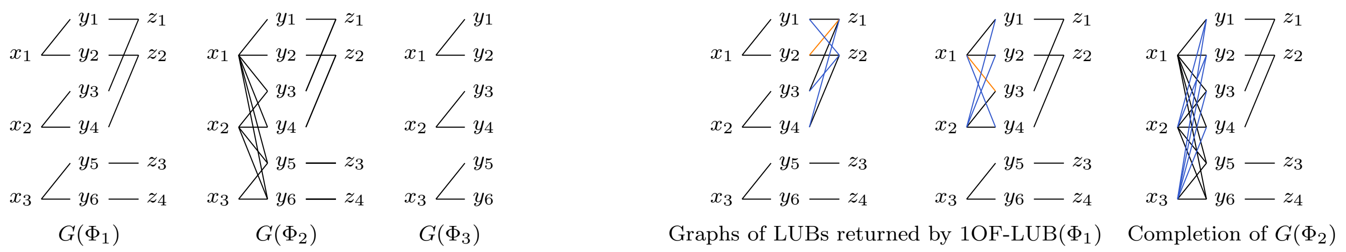
$G(\Phi_1)$
$G(\Phi_2)$
$G(\Phi_3)$
Graphs of LUBs returned by 1OF-LUB(Φ_1)
Completion of $G(\Phi_2)$

Figure 2: Graphs of Φ_1, Φ_2, Φ_3 from Figure 1. Edges (x_i, z_j) are not shown to avoid clutter, but can be inferred here since x_i is connected to y_l's that are connected to z_j's. The fourth and fifth graphs represent LUBs returned by Algorithm 3; orange and blue edges are added due to the alignment and completion steps, respectively. The sixth graph depicts the completion of $G(\Phi_2)$.

these graphs and their complements. Before any complementation, the formula is obtained as the disjunction of the subformulas for the connected components. With each complementation, we alternate conjunctions and disjunctions. In case of a node, we return its literal. If G represents an 1OF formula, then this procedure necessarily finds it.

Example 10. Consider the graph G that is the third from left in Figure 2. It has three components: C_1, C_2, C_3. Then, $\rho(G) = \rho(C_1) \vee \rho(C_2) \vee \rho(C_3)$. We next illustrate for C_1, the same applies to the other two components. We complement it, and obtain two components: C_{11} consisting of one node x_1 and C_{12} consisting of two connected nodes y_1 and y_2. Then, $\rho(C_1) = x_1 \wedge \rho(C_{12})$. We complement C_{12} and obtain two components, one with the node y_1 and the other with the node y_2. Hence, $\rho(C_{12}) = y_1 \vee y_2$. The 1OF formula represented by $G(\Phi_3)$ is $x_1(y_1 \vee y_2) \vee x_2(y_3 \vee y_4) \vee x_3(y_5 \vee y_6)$.

For the rightmost graph in Figure 2, we complement it and obtain two connected components: One component consists of a clique of three nodes x_1, x_2, x_3, and the other consists of all the remaining nodes. The obtained formula is thus a conjunction of the formulas of these two components. After several decomposition steps, we obtain

$$(x_1 \vee x_2 \vee x_3)[z_1(y_1 \vee y_3) \vee z_2(y_2 \vee y_4) \vee z_3 y_5 \vee z_4 y_6]. \Box$$

We next give an alternative graph-based characterisation of k-partite 1OF formulas. This is used in the next sections to create optimal 1OF bounds and is based on two graph notions: alignment and completeness. A bipartite graph (X, Y, E) is complete if $E = X \times Y$.

Definition 10. Two bipartite graphs $G = (V_1, V_2, E)$ and $G' = (V_1', V_2', E')$ are *aligned* if for all pairs (V_i, V_j'), $1 \leq i, j \leq 2$, it holds that $V_i \cap V_j' = \emptyset$ or $V_i \subseteq V_j'$ or $V_j' \subset V_i$. $\Box$

Our characterisation of k-partite 1OFs is as follows:

Lemma 3. *Let $\mathcal{B}$ be the set of projection graphs of a unate k-partite formula. The set of connected components of the bipartite graphs in $\mathcal{B}$ are complete and pairwise aligned if and only if the formula represented by $\mathcal{B}$ is in 1OF.* $\Box$

Since a k-partite formula Φ is in general not normal, its graph may represent a different formula. The next lemma shows that the represented formula is a k-partite upper bound of Φ and that there is no smaller normal k-partite upper bound. This implies that the graph only represents clauses that are necessarily in every k-partite 1OF upper bound.

Lemma 4. *Let Φ be a k-partite formula, $\mathcal{B}$ its set of projection graphs, and Φ_u the formula represented by $\mathcal{B}$. Then Φ_u is k-partite, $\mathcal{M}(\Phi) \subseteq \mathcal{M}(\Phi_u)$, and there is no normal k-partite formula Φ_u' such that $\mathcal{M}(\Phi) \subseteq \mathcal{M}(\Phi_u') \subset \mathcal{M}(\Phi_u)$.*

The following lemma states that in order to find optimal 1OF bounds for a unate k-partite formula Φ, it is sufficient to remove clauses from Φ or add clauses to Φ.

Lemma 5. *Let Ψ be a GLB (LUB) for a unate k-partite formula Φ with respect to k-partite 1OFs. Then Ψ is GLB (LUB) for Φ with respect to general 1OFs.* $\Box$

Both Lemmata 3 and 5 are key ingredients for finding optimal 1OF bounds for unate k-partite formulas.

5.1 1OF Least Upper Bounds

Algorithm 3 enumerates optimal k-partite upper bounds using an approach that closely follows Lemma 3. We first *complete* all connected components of bipartite projection graphs of the input formula. Then, we *align* components from different projection graphs that have overlapping, but incomparable sets of nodes, by introducing new edges across these components. Whereas there is one possible completion for these graphs, there may be two possible alignments for any pair of components.

While the completion step implements the canonical notion of a complete bipartite graph, the alignment step needs further explanation. If two components $C = (V, W, E)$ and $C' = (V', W', E')$ are non-aligned, then at least one pair of variable sets $(V, V'), (V, W'), (W, V'), (W, W')$ — assume wlog it is (V, V') — must be incomparable, i.e. satisfy $V \cap V' \neq \emptyset$ and $V \not\subseteq V'$ and $V \not\supset V'$. We align the components C, C' by enlarging C' such that $V \subseteq V'$. This is achieved by adding edges $(V - V') \times W'$ to the projection graph of C' as shown in the *Align* procedure in Algorithm 3. All these edges are necessary for alignment; further edges could be added but they would violate the optimality of our construction of least upper bounds.

Example 11. The graph $G(\Phi_1)$ from Figure 2 has three bipartite projection graphs $\mathcal{B}(\Phi) = \{B_{xy}, B_{xz}, B_{yz}\}$. Consider the sets of variables $V_1 = \{y_1, y_2\}$, $V_2 = \{y_3, y_4\}$, $V_3 = \{y_1, y_3\}$, and $V_4 = \{y_2, y_4\}$. The first two sets and the last two sets belong to different components of the projections graphs B_{xy} and B_{yz}, respectively. Any of V_1 and V_2 is incomparable with any of V_3 and V_4, that is, they overlap yet none is included in the other. This means that their components are not aligned. Diagrams four and five in Figure 2 show two possible alignment and completion steps for $G(\Phi_1)$. The edge $y_1 - z_2$ (orange) in the fourth diagram aligns node sets V_1 and V_3 by extending the component containing nodes $V_3 = \{y_1, y_3\}$ in B_{yz} by node y_2, leading to $V_1 \subseteq \{y_1, y_2, y_3\}$. The edges $y_1 - z2$, $y_3 - z_2$, and $y_4 - z_2$ (blue) are due to the subsequent completion step in the projection graph B_{yz}.

The projection graph B_{xy} in the graph $G(\Phi_2)$ has one component only. This component is not complete, since, for instance, $(x_3, y_1) \notin B_{xy}$. The rightmost diagram in Figure 2 shows $G(\Phi_2)$ after completion. The projection graphs in the latter graph are already aligned: Indeed, the variable set $\{y_1, \ldots, y_6\}$ of the large component includes the variable sets $\{y_1, y_3\}$, $\{y_2, y_4\}$, and $\{y_5, y_6\}$ of the other components. □

Both steps, completion and alignment, may introduce new clauses that are necessary to obtain a 1OF formula. They detect P_4 paths and add edges between their nodes that correspond to new clauses of arity k. When these steps are done, the graph is P_4-free and defines a 1OF formula.

We are now ready to state the main result of this section.

THEOREM 3. *Algorithm 3 computes k-partite 1OF least upper bounds of a unate k-partite formula.* □

Soundness follows by Lemmata 3 and 5 and the fact that by completion and alignment no unnecessary edges are added to the graph (in other words, no unnecessary clause is added to the input formula). In particular, note that according to Lemma 5, bounds computed by Algorithm 3 are optimal even with respect to the 1OF language, and not only to the language of k-partite 1OFs.

Example 12. The aligned and completed graphs of $G(\Phi_1)$ given in Figure 2 correspond to the 1OF formulas:

$$1: \quad (x_1 \lor x_2)[z_1(y_1 \lor y_3) \lor z_2(y_2 \lor y_4)] \lor x_3(y_5z_3 \lor y_6z_4)$$
$$2: \quad [x_1(y_1 \lor y_2) \lor x_2(y_3 \lor y_4)](z_1 \lor z_2) \lor x_3(y_5z_3 \lor y_6z_4)$$

These two formulas are the only 3-partite LUBs for Φ_1.

The completion of $G(\Phi_2)$ given in Figure 2 corresponds to the only 1OF 3-partite LUB formula for Φ_2:

$$\Phi_U^2 = (x_1 \lor x_2 \lor x_3)[z_1(y_1 \lor y_3) \lor z_2(y_2 \lor y_4) \lor z_3y_5 \lor z_4y_6]$$

Example 10 shows how to compute the 1OF formula of the graph obtained after completing $G(\Phi_2)$. □

THEOREM 4. *Algorithm 3 enumerates 1OF least upper bounds for a k-partite unate DNF formula with polynomial delay.* □

The graph of a k-partite unate DNF formula can be constructed in time polynomial in k, in the size and in the number of variables of the formula, and has $O(k^2)$ bipartite subgraphs. Each iteration of the *foreach*-loop outputs a distinct LUB, since every iteration fixes a projection graph B that is never aligned; two LUBs produced by different runs are thus incomparable. It remains to be shown that any *foreach*-iteration takes at most polynomial time. The number of alignment and completion steps — hence the number of *while*-iterations — required is bounded by the maximal number of edges in the graph, $O(n^2)$. Each alignment and completion step in a component of a projection graph costs at most quadratic time in the number of its nodes.

Enumeration of all k-partite 1OF LUBs. Algorithm 3 does not enumerate *all* k-partite 1OF LUBs of a given formula. This is due to the restriction that for each projection graph B that has a non-aligned component only one LUB is constructed. By removing the *foreach*-statement from the algorithm and implementing the choice in the *while*-statement to be non-deterministic, this restriction can be lifted to obtain a non-deterministic algorithm that finds all k-partite 1OF LUBs. To obtain a deterministic version of this algorithm, we iterate and branch over all such choices. Since different branches may find the same LUB several times, this algorithm does not enumerate LUBs with polynomial delay. Indeed, this extended algorithm has exponential time complexity, since (i) we can choose one of two possible alignments for each pair of non-aligned projection graphs, and (ii) the number of alignments of a k-partite graph can be exponential in the size of the graph.

```
1OF-GLB ($k$-partite unate formula $\Phi$)
outputs all 1OF GLBs of $\Phi$ that are $k$-partite

$\Gamma \leftarrow \emptyset$    // $\Gamma$ is the set of all GLBs seen so far
foreach $n$ from $|\Phi|$ downto 1 do
    foreach $\Phi' \subseteq \Phi$ such that $|\Phi'| = n$ do
        if $\Phi'$ is in 1OF and $\nexists \Phi_0 \in \Gamma$ such that $\Phi' \subseteq \Phi_0$
        then
            Output $\rho(\Phi')$
            if $\Phi' = \Phi$ then
                return
            $\Gamma \leftarrow \Gamma \cup \{\Phi'\}$
```

Algorithm 4: Finding 1OF GLBs of a k-partite unate formula.

PROPOSITION 9. *The graph of the 3-partite formula* $\Phi_n^3 = \bigvee_{i=1}^{n} \left(x_iy_{4i-3}z_{4i-3} \lor x_iy_{4i-2}z_{4i-2} \lor x_{i+1}y_{4i-1}z_{4i-3} \lor x_iy_{4i}z_{4i-2}\right)$ *has size* $4n$ *and* 2^n *alignments.* □

The graph in Proposition 9 consists of n unconnected copies of the subgraph represented by the upper part U of the graph $G(\Phi_1)$ in Figure 2. There are exponentially many alignments since U has two alignments and the n copies of U are unconnected.

Remark 2. Algorithm 3 can also be used to compute k-partite 1OF upper bounds for formulas that represent lineage of conjunctive queries *with self-joins.* Those formulas are similar to k-partite ones, except that they are not over disjoint variable sets $V_1, \ldots, V_k$, but some V_i, V_j can be identical. To obtain bounds in this case, first drop a minimal number of variables from clauses such that each clause contains at most one variable from each distinct variable set. Subsequently, Algorithm 3 can be applied to this smaller formula. An important direction of future work is to study the case of optimal bounds for such formulas as well as unrestricted unate formulas. □

5.2 1OF Greatest Lower Bounds

For a given k-partite unate formula Φ, Algorithm 4 computes k-partite 1OF greatest lower bounds. It iterates over all subformulas Φ' representing subsets of Φ starting with the largest subformulas first. If Φ' is a 1OF formula and is not included in any bounds previously found by the algorithm, then Φ' is reported as a greatest lower bound.

PROPOSITION 10. *Algorithm 4 computes all k-partite 1OF greatest lower bounds of a unate k-partite formula.* □

Since it only checks for subsets of the input formula Φ, the algorithm reports lower bounds that have arity k, which is sufficient according to Lemma 5. The 1OF lower bounds computed by the algorithm are optimal, since we check that there is no larger 1OF lower bound. We do this by iterating over the larger subformulas before the smaller subformulas, and also over Φ, and memorizing all GLBs encountered so far (in Γ). The algorithm is complete since it iterates over the entire space of possible k-partite lower bounds.

Example 13. Formula Φ_1 in Figure 1 has four 1OF GLBs:

$$\begin{aligned}
1: &\quad (x_1y_1 \lor x_2y_3)z_1 \lor x_3(y_5z_3 \lor y_6z_4)\\
2: &\quad x_1y_1z_1 \lor x_2y_4z_2 \lor x_3(y_5z_3 \lor y_6z_4)\\
3: &\quad (x_1y_2 \lor x_2y_4)z_2 \lor x_3(y_5z_3 \lor y_6z_4).\\
4: &\quad x_1y_2z_2 \lor x_2y_3z_1 \lor x_3(y_5z_3 \lor y_6z_4).\Box
\end{aligned}$$

Algorithm 4 needs exponential time to report all k-partite 1OF GLBs of a unate DNF formula. This is partly due to the exponential number of such lower bounds. The problem of enumerating 1OF greatest lower bounds with polynomial delay remains open.

Proposition 11. *The unate bipartite DNF formula*

$$\Phi = (x_1y_1 \lor x_1y_2 \lor x_2y_2) \lor \cdots \lor (x_{2n-1}y_{2n-1} \lor x_{2n-1}y_{2n} \lor x_{2n}y_{2n})$$

has $3n$ clauses and 3^n 1OF bipartite greatest lower bounds.

Any formula consisting of either $x_{2i-1}(y_{2i-1} \lor y_{2i})$, or $(x_{2i-1} \lor x_{2i})y_{2i}$, or $(x_{2i-1}y_{2i-1} \lor x_{2i}y_{2i})$ for all $1 \le i \le n$ is a 1OF bipartite greatest lower bound of Φ. $\Box$

6. APPROXIMATION BY QUERIES

In this section we show how to derive *queries that compute bounds* for the lineage of conjunctive queries without self-joins on databases in which each tuple is annotated with a fresh Boolean variable. In case of random variables, such databases are tuple-independent probabilistic databases. Bounds for the lineage are expressed in the languages of iDNF or 1OF, as discussed in the previous sections.

Our motivation behind this study is twofold. Firstly, bounds that are expressible by queries can be computed in polynomial time using relational database engines. This is essential for probabilistic databases, where query evaluation is #P-hard in general since it involves probability computation of arbitrary propositional formulas. Secondly, we would like to understand whether queries with polynomial *data* complexity on classical relational databases can be approximated by bound queries that have polynomial *combined* complexity and have answers representable in the iDNF or 1OF propositional languages.

6.1 Expressing Bound Queries

The main challenge in expressing bounds using queries is the encoding (i) of the completion and alignment steps for 1OF upper bounds, and (ii) of the choice of a maximal subset of clauses that is still in iDNF or 1OF for lower bounds. Deriving queries that express iDNF upper bound computation is subject to future work.

We first discuss the case of 1OF bounds. For a given conjunctive query Q, its lineage is already in 1OF if Q is hierarchical and has no self-joins [17]. By Definition 3, the hierarchical property is violated by relation symbols with several query variables that also appear in other distinct relation symbols. For instance, relation S in queries Q_1, Q_2,

```
Query-UB (Conjunctive Query without Self-Joins $Q$)
outputs upper bound queries for $Q$

foreach non-head query variables $A, B$ such that not
$(s(A) \cap s(B) = \emptyset$ or $s(A) \subseteq s(B)$ or $s(A) \supset s(B))$ do
    foreach relation $S \in (s(A) \cap s(B))$ do
        //Step 1: complete $S$ on columns $(A, B)$
        replace $S$ by $\mathbf{trans}_Q(S, A, B)$ in $Q$

        //Step 2: align $S$ on columns $(A, B)$
        $\alpha_A(A, B, \bar{X})$:-$S(A, B', \bar{X}), S^d(A', B, \bar{X}), S^d(A', B', \bar{X})$
        $\alpha_B(A, B, \bar{X})$:-$S(A', B, \bar{X}), S^d(A, B', \bar{X}), S^d(A', B', \bar{X})$

        choose one of the following:
        1. replace $S$ by $\alpha_A$ in $Q$
        2. replace $S$ by $\alpha_B$ in $Q$
output $Q$
```

Algorithm 5: Deriving upper bound queries.

and Q_3 of Figure 1 violates the hierarchical property. In case of Q_1, the relation S can be constructed such that it pairs arbitrary tuples from relations R and T, and in particular arbitrary annotations from the two relations. The lineage of such queries cannot thus be in 1OF in general, though it may be in 1OF for particular extensions or reductions of such violating relations. The key insight is that one particular case of extensions or reductions of violating relations is by means of completion and alignment of their content.

Algorithm 5 extends Q with completion and alignment steps necessary for computing 1OF upper bounds, as detailed in Section 5.1. It first detects the binary projections of violating relations S that have incomplete and/or misaligned connected components. This detection is based on the hierarchical property of conjunctive queries as given in Definition 3, which corresponds precisely to the alignment check used by Algorithm 3.

The completion step requires transitive closure on the bipartite graph G_{AB} represented by the projection $S(A, B)$: For each tuple t from a relation joined with S via an equality or inequality condition on A and that has an A-value a, we introduce new tuples in S. These tuples allow for combinations of t with B-values that are annotated by their variables in S and that appear with A-values from the same connected component of G_{AB} with value a. We denote this closure by $\mathbf{trans}_Q(S, A, B)$ in Algorithm 5.

Example 14. Consider the query Q:-$R(A), S(A, B), T(B)$, where the relations are defined as follows:

R	A
x_1	a_1
x_2	a_2

S	A	B
y_1	a_1	b_1
y_2	a_1	b_2
y_3	a_1	b_3
y_4	a_2	b_1
y_5	a_2	b_4

T	B
z_1	b_1
z_2	b_2
z_3	b_3
z_4	b_4

S'	A	B
y_4	a_1	b_1
y_5	a_1	b_4
y_1	a_2	b_1
y_2	a_2	b_2
y_3	a_2	b_3

The completion of the violating relation S leads to the addition of the tuples S' to S. The lineage of the upper bound query Q^u:-$R(A), \mathbf{trans}_Q(S, A, B), T(B)$ is the 1OF formula $(x_1 \lor x_2)[(y_1 \lor y_4)z_1 \lor y_2z_2 \lor y_3z_3 \lor y_5z_4]$. This is indeed an upper bound of the original lineage $x_1(y_1z_1 \lor y_2z_2 \lor y_3z_3) \lor x_2(y_4z_1 \lor y_5z_4)$, which is not in 1OF. $\Box$

After completion, the alignment step can add tuples to S so that all variables associated with either A or B are now

```
Query-LB (Conjunctive Query without Self-Joins $Q$)
outputs lower bound queries for $Q$

choose an order $<_Q$ on the query variables of $Q$
foreach non-head query variables $A, B$ such that not
($s(A) \cap s(B) = \emptyset$ or $s(A) \subseteq s(B)$ or $s(A) \supset s(B)$) do
    1. In case of 1OF bounds:
    if $A <_Q B$ then
        $f \leftarrow$ functional dependency ($A \rightarrow B$)
    else
        $f \leftarrow$ functional dependency ($B \rightarrow A$)
    2. In case of iDNF bounds:
    $f \leftarrow$ functional dependencies ($A \rightarrow B, B \rightarrow A$)
    foreach relation $S$ in $s(A) \cap s(B)$ do
        replace $S$ by $repair_f(S)$ in $Q$
output $Q$
```

Algorithm 6: Deriving lower bound queries.

also associated with the other B or A values, respectively, in the same connected component G_{AB}. We denote alignment by α_v in the algorithm; S^d is relation S without variables and v denotes our choice of aligning on either A or B. The alignment step can be encoded as a conjunctive query.

Example 15. In query Q_1 from Figure 1, S is the only violating relation. This relation is already complete. There are two possible bound queries: $Q_1^{u_1}$:-$R(A), \alpha_A(A,B), T(B)$ and $Q_1^{u_2}$:-$R(A), \alpha_B(A,B), T(B)$, where α is defined as given, where in addition S is replaced by $\mathbf{trans}_{Q_1}(S,A,B)$. After the alignment α_A, the following tuples are added to S: $y_1(1,2), y_2(1,1)$, $y_3(2,2)$, $y_4(2,1)$, $y_5(3,4)$, and $y_6(3,3)$. With this extended relation S, the query would produce the lineage $[x_1(y_1 \vee y_2) \vee x_2(y_3 \vee y_4)](z_1 \vee z_2) \vee x_3(y_5 \vee y_6)(z_3 \vee z_4)$. This lineage is an 1OF upper bound of the original lineage.□

THEOREM 5. *Given a conjunctive query without self-joins Q, Algorithm 5 constructs queries Q_U with a transitive closure construct such that:*

- *The size of Q_U is at most quadratic in the size of Q,*
- *For every relational database D, $Q(D) \subseteq Q_U(D)$,*
- *For every annotated database D, the lineage of $Q_U(D)$ is a $|Q|$-partite 1OF formula, and*
- *For every probabilistic database D and tuple $t \in Q(D)$, $P(t \in Q(D)) \leq P(t \in Q_U(D))$ and, given $Q_U(D)$, $P(t \in Q_U(D))$ can be computed in polynomial time.* □

This result draws on the connection between Algorithms 5 and 3, since the former only provides query encodings of the completion and alignment steps of the latter. It can be further shown that the completion step is redundant when all joins are key-foreign key joins. If in addition these joins build a hierarchy based on inclusion of the query variables involved in each join, then the query is hierarchical and the alignment step is also not needed, since then the lineage is guaranteed to be in 1OF. Alternative upper bound queries without the transitive closure construct can be constructed by dropping one of the join conditions for each violating relation in the query.

Theorem 5 also holds for input queries with inequality joins ($<$). In case of queries with self-joins, we can obtain upper bounds by first dropping all but one occurrence of each relation symbol in the query, and then use Algorithm 5.

To express lower bound computation using queries, we need to resolve incompletion and misalignment of graph components of many-to-many relations by dropping tuples and hence clauses from the lineage to be approximated. This can be achieved by non-deterministically choosing maximal functional bipartite subgraphs of many-to-many binary relations, and as such, it requires a choice operator, whose expressiveness is beyond first order [20]. A similar operator has been proposed in the context of uncertain databases for constructing all possible (maximal) *repairs* of a relation under given functional dependencies [1]. Algorithm 6 uses a repair operator to choose a maximal subset of a relation that satisfies a given set of functional dependencies. An alternative approach to the choice operator is to extend the language of first-order queries with a new binary order $<$ operator, and consider an arbitrary linear ordering of the universe. Linearly ordered finite domains provide a first-order definable choice function, namely the one that always picks $<$-least elements [20].

THEOREM 6. *Given a conjunctive query without self-joins Q, Algorithm 6 constructs queries Q_L with a repair construct such that:*

- *The size of Q_L is at most quadratic in the size of Q,*
- *For every relational database D, $Q_L(D) \subseteq Q(D)$,*
- *For every annotated database D, the lineage of $Q_L(D)$ is a $|Q|$-partite 1OF or iDNF formula, and*
- *For every probabilistic database D and tuple $t \in Q(D)$, $P(t \in Q_L(D)) \leq P(t \in Q(D))$ and, given $Q_L(D)$, $P(t \in Q_L(D))$ can be computed in polynomial time.* □

By enforcing a functional dependency $A \rightarrow B$ on each violating relation S in the query, we turn the bipartite graph of $S(A,B)$ into the graph of a function, and drop clauses from the lineage of the original query that forbid factorisation in 1OF. In the iDNF case, we enforce a two-way dependency $A \leftrightarrow B$ on each violating relation S, hence each variable annotating a tuple in S can only occur in one clause in the lower bound lineage. In the 1OF case, the constructed query turns the database into a chain of one-to-many joined relations. All connected components of the projection graphs of the lineage graph are then necessarily complete and aligned, hence the lineage is in 1OF.

Example 16. We enforce any of the functional dependencies $A \rightarrow B$ or $B \rightarrow A$ on relation S in the query Q_1 from Figure 1. The number of maximal functional subgraphs of S is eight in the first case and four in the second case. In the former case, the 1OF lower bound query would be $Q_1^{l_1}$:-$R(A), repair_{A\rightarrow B}(S(A,B)), T(B)$. If we choose the first tuples for each A-value in S, we obtain the lineage $(x_1y_1 \vee x_2y_3)z_1 \vee x_3y_5z_3$.

The query $Q_1^{l_2}$:-$R(A), repair_{A\leftrightarrow B}(S(A,B)), T(B)$ is the iDNF lower bound query for Q_1. If we choose the first, fourth, and sixth tuples in S, we obtain the lineage $x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_5z_3$. The iDNF bound query is optimal since its bound is optimal. The 1OF bound query is not optimal, Example 13 lists the 1OF greatest lower bounds for Q_1. □

As exemplified above, the 1OF lower bound queries constructed by Algorithm 6 are usually not optimal. To obtain optimality, we would need a more involved version of the repair operator that can enforce any of the two functional dependencies independently for each of the connected components of the bipartite graph of $S(A,B)$.

6.2 iDNF and 1OF Representation Systems

Since iDNF and 1OF formulas representing the lineage of bound queries have sizes linear in the number of their variables, a natural question is whether bound queries can be evaluated more efficiently than the queries they approximate. The initial answer is "no", since there are bound queries, such as the product of n relations, that produce answers of size exponential in n. The structure of answers to bound queries, however, follows the regular structure of 1OF or iDNF languages. This motivates the use of these languages as succinct representations of query answers.

Definition 11. Let Q be a 1OF bound query, D be an annotated database, and Φ be the 1OF lineage of the query answer $Q(D)$. The *1OF representation* of $Q(D)$ is $(\Phi_e, \mathcal{S})$, where $\mathcal{S}$ is the schema of the query answer and Φ_e is Φ extended by inserting at each variable x occurring in Φ the input tuple t annotated with x together with the schema of t. The case of iDNF is defined analogously. $\Box$

Example 17. Consider the lineage of the 1OF upper bound query Q_1^{u1} on database D from Example 15: $[x_1(y_1 \vee y_2) \vee x_2(y_3 \vee y_4)](z_1 \vee z_2) \vee x_3(y_5 \vee y_6)(z_3 \vee z_4)$. The 1OF representation of $Q_1^{u1}(D)$ is $(\Phi_e, \mathcal{S})$ with $\mathcal{S} = \emptyset$ and

$$\begin{aligned}\Phi_e = \Big[&x_1\langle 1\rangle(y_1\langle 1,1\rangle \vee y_2\langle 1,2\rangle)\vee \\ &x_2\langle 2\rangle(y_3\langle 2,1\rangle \vee y_4\langle 2,2\rangle)\Big](z_1\langle 1\rangle \vee z_2\langle 2\rangle) \\ &\vee \Big[x_3\langle 3\rangle(y_5\langle 3,3\rangle \vee y_6\langle 3,4\rangle)\Big](z_3\langle 3\rangle \vee z_4\langle 4\rangle).\Box\end{aligned}$$

This representation is closed under queries with projection, selection, and product operators, but not under queries with joins and projection. In contrast to classical DNF representations of query answers (equivalent to a list of answer tuples), the 1OF representation is more succinct and enables lower complexity of query evaluation for 1OF bound queries.

THEOREM 7. *Given a bound query Q as constructed by Algorithms 5 and 6 and an annotated database D, the 1OF or iDNF representation of $Q(D)$ needs space $O(|D|)$ and can be computed in polynomial combined complexity.*

Given an 1OF or iDNF representation of $Q(D)$, the answers to $Q(D)$ can be enumerated with polynomial delay. $\Box$

7. CONCLUSION AND FUTURE WORK

This paper introduces a novel approach to query approximation in probabilistic and relational databases via two tractable propositional languages. We show equivalences between semantic and syntactic characterisations of optimal bounds; while the former provide an intuitive understanding of optimality, the latter are more easily accessible to efficient algorithmic treatment. We propose two methods for the computation of optimal bounds, one based on the syntax of lineage formulas, and one based on query rewriting.

We see several promising directions for future work. We plan to integrate the approximation framework into our query engine for probabilistic databases called SPROUT. The application of our results to the relational case suggests a novel paradigm to the representation and computation of query answers. A promising direction is the investigation of queries whose answers are k-readable, that is, answers that can be represented as formulas where each input tuple occurs at most k times, where k is a polynomial in the sizes of the database and the query. The problem of enumerating 1OF greatest lower bounds with polynomial delay is still open.

Acknowledgments. This research was funded by the FP7 ERC grant agreement FOX number FP7-ICT-233599.

8. REFERENCES

[1] L. Antova, C. Koch, and D. Olteanu. "From Complete to Incomplete Information and Back". In *SIGMOD*, 2007.

[2] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7), 2009.

[3] N. Dalvi and D. Suciu. "Efficient Query Evaluation on Probabilistic Databases". *VLDB Journal*, 16(4), 2007.

[4] A. Darwiche and P. Marquis. "A knowlege compilation map". *Journal of AI Research*, 17, 2002.

[5] K. Elbassioni, K. Makino, and I. Rauf. On the Readability of Monotone Boolean Formulae. In *COCOON*, 2009.

[6] R. Fink and D. Olteanu. On the Optimal Approximation of Queries Using Tractable Propositional Languages. [Extended version, available on author's homepage]. 2011.

[7] R. Fink, D. Olteanu, and S. Rath. "Providing Support for Full Relational Algebra Queries in Probabilistic Databases". In *ICDE*, 2011.

[8] M. Garofalakis and P. Gibbon. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[9] W. Gatterbauer, A. K. Jha, and D. Suciu. Dissociation and propagation for efficient query evaluation over probabilistic databases. TR UW-CSE-10-04-01, U. Washington, 2010.

[10] M. Golumbic, A. Mintza, and U. Rotics. "Read-Once Functions Revisited and the Readability Number of a Boolean Function". In *Int. Colloq. on Graph Theory*, 2005.

[11] C. P. Gomes, A. Sabharwal, and B. Selman. *Handbook of Satisfiability*, chapter Model Counting. IOS Press, 2009.

[12] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[13] V. Gurvich. Criteria for repetition-freeness of functions in the algebra of logic. In *Soviet math. dolk.,43(3)*, 1991.

[14] T. Imielinski and W. Lipski. "Incomplete information in relational databases". *Journal of ACM*, **31**(4), 1984.

[15] D. Johnson, C. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 1988.

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

[17] D. Olteanu and J. Huang. "Using OBDDs for Efficient Query Evaluation on Probabilistic Databases". In *SUM*, 2008.

[18] D. Olteanu and J. Huang. "Secondary-Storage Confidence Computation for Conjunctive Queries with Inequalities". In *SIGMOD*, 2009.

[19] D. Olteanu, J. Huang, and C. Koch. "Approximate Confidence Computation for Probabilistic Databases". In *ICDE*, 2010.

[20] M. Otto. Epsilon Logic is More Expressive than First-Order Logic over Finite Structures. *J. Symb. Log.*, 65(4), 2000.

[21] J. Pe'er and R. Y. Pinter. Minimal decomposition of boolean functions using non-repeating literal trees. In *IFIP Workshop on Logic and Architecture Synthesis*, 1995.

[22] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. of the ACM*, 27(4), 1980.

[23] B. Selman. Knowledge compilation and theory approximation. *J. of the ACM*, 43(2), 1996.

[24] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. In *VLDB*, 2010.

[25] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM J. Comput.*, 6(3), 1977.

[26] S. Vadhan. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM J. Comput.*, 32(2), 2001.