Top-k Query Processing in Probabilistic Databases with Non-Materialized Views

Maximilian Dylla [#1], Iris Miliaraki [#2], Martin Theobald [†3]
[#] *Max Planck Institute for Informatics*
Campus E1.4, 66123 Saarbrücken, Germany
{[1]mdylla,[2]miliaraki}@mpi-inf.mpg.de

[†] *University of Antwerp*
Middelheimlaan 1, 2020 Antwerp, Belgium
[3] martin.theobald@ua.ac.be

***Abstract*—We investigate a novel approach of computing confidence bounds for top-k ranking queries in probabilistic databases with *non-materialized views*. Unlike related approaches, we present an exact pruning algorithm for finding the top-ranked query answers according to their marginal probabilities *without* the need to first materialize all answer candidates via the views. Specifically, we consider conjunctive queries over multiple levels of select-project-join views, the latter of which are cast into Datalog rules which we ground in a top-down fashion directly at query processing time. To our knowledge, this work is the first to address *integrated data and confidence computations* for intensional query evaluations in the context of probabilistic databases by considering confidence bounds over *first-order lineage* formulas. We extend our query processing techniques by a tool-suite of *scheduling strategies* based on selectivity estimation and the expected impact on confidence bounds. Further extensions to our query processing strategies include improved top-k bounds in the case when *sorted relations* are available as input, as well as the consideration of *recursive rules*. Experiments with large datasets demonstrate significant runtime improvements of our approach compared to both exact and sampling-based top-k methods over probabilistic data.**

I. Introduction

Managing uncertain data via probabilistic databases (PDBs) has evolved as an established field of research in recent years, with a plethora of applications ranging from scientific data management, sensor networks, data integration, to knowledge management systems [1]. Despite the polynomial runtime complexity for the data computation step involved in finding probabilistic answer candidates, confidence computations for these answers are known to be $\#\mathcal{P}$-hard already for fairly simple select-project-join (SPJ) queries [2], [3]. Thus, efficient strategies for confidence computation and early pruning of low-confidence query answers remain a key challenge for the scalable management of uncertain data.

Recent work on efficient confidence computations in PDBs has addressed this problem mainly from two ends, namely by restricting the class of queries, i.e., by focusing on *safe query plans* [2], or by considering a specific class of tuple-dependencies, commonly referred to as *read-once functions* [4]. Intuitively, safe query plans denote a class of queries for which confidence computations can directly be coupled with the relational operators and thus be performed by an extensional query plan [1]. On the other hand, read-once formulas denote a class of propositional lineage formulas which can be factorized (in polynomial time) into a form where every variable representing a database tuple appears at most once, thus again permitting efficient confidence computations.

While safe plans focus on the characteristics of the query structure, and read-once formulas focus on the logical dependencies among individual data objects, top-k style pruning approaches, which are the subject of this work, have recently been proposed as an alternative way to address confidence computations in PDBs [5], [6], [7]. These approaches aim to efficiently identify the top-k most probable answers, using lower and upper bounds for their marginal probabilities, without the need to compute the exact probabilities of these answers. Suciu et al. [5], [7] addressed this by approximating the probabilities of the top-k answers using Monte-Carlo-style sampling techniques. Olteanu and Wen [6] have further developed their idea of decomposing propositional formulas for deriving confidence bounds based on partially expanded, ordered binary decision diagrams (OBDDs) [8], which can again be exploited by top-k algorithms for early candidate pruning. In particular the latter top-k algorithm [6] can effectively circumvent the need for exact confidence computations and can still—in many cases—return the top-ranked query answers in an exact way. However, as opposed to top-k approaches in traditional DBs [9], [10], none of the former saves upon the data computation step needed to find the answer candidates. Thus, extensive data materialization is required for queries with multiple nested subqueries or over multiple levels of potentially convoluted views.

In this paper, we specifically focus on the case when *views are not materialized*. We are aiming to identify the top-ranked query answers, based on their marginal probabilities, before all input tuples that would be needed to compute the query answers in an exhaustive way have been seen by the query processor. Following the line of works on intensional query evaluation [1], [11], [12], [13], we employ lineage formulas to capture the logical dependencies between query answers and the input tuples that were employed to derive these answers via the views. In contrast to all lineage models known to us, which consider lineage as purely propositional formulas [1], [14], and where each formula represents a single query answer, we more generally introduce *first-order lineage*

Directed

	Director	*Movie*	p
t_1	Coppola	ApocalypseNow	0.8
t_2	Coppola	Godfather	0.9
t_3	Tarantino	PulpFiction	0.7

ActedIn

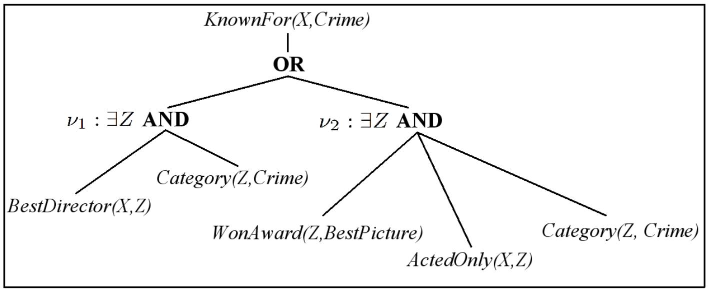
	Actor	*Movie*	p
t_4	Brando	ApocalypseNow	0.6
t_5	Pacino	Godfather	0.3
t_6	Tarantino	PulpFiction	0.4

WonAward

	Movie	*Award*	p
t_7	ApocalypseNow	BestScript	0.3
t_8	Godfather	BestDirector	0.8
t_9	Godfather	BestPicture	0.4
t_{10}	PulpFiction	BestPicture	0.9

Category

	Movie	*Category*	p
t_{11}	ApocalypseNow	War	0.9
t_{12}	Godfather	Crime	0.5
t_{13}	PulpFiction	Crime	0.9
t_{14}	Inception	Drama	0.6



$\nu_1 : \forall X, Y \ \ \mathit{KnownFor}(X, Y) \ \ \text{:-}\ \exists Z \ \ \mathit{BestDirector}(X, Z), \mathit{Category}(Z, Y)$

$\nu_2 : \forall X, Y \ \ \mathit{KnownFor}(X, Y) \ \ \text{:-}\ \exists Z \ \ \mathit{WonAward}(Z, \mathit{BestPicture}), \mathit{ActedOnly}(X, Z), \mathit{Category}(Z, Y)$

$\nu_3 : \forall X, Y \ \ \mathit{BestDirector}(X, Y) \ \text{:-}\ \mathit{Directed}(X, Y), \mathit{WonAward}(Y, \mathit{BestDirector})$

$\nu_4 : \forall X, Y \ \ \mathit{ActedOnly}(X, Y) \ \ \text{:-}\ \mathit{ActedIn}(X, Y), \neg \mathit{Directed}(X, Y)$

Fig. 1. A Movie Database and a Partially Grounded Lineage Formula for the Query $\mathit{KnownFor}(X, \mathit{Crime})$

formulas, where each formula may represent an entire *set of query answers*. Our main observation is that each intermediate step of query processing can be unambiguously described and thus be captured by such a first-order lineage formula, which is our key for combining data and confidence computations in a probabilistic database setting. We illustrate this by the following example.

Example 1: Fig. 1 depicts a probabilistic database that consists of the extensional relations *Directed*, *ActedIn*, *Category*, and *WonAward*, as well as views ν_1–ν_4 in Datalog notation which define the intensional relations *KnownFor*, *BestDirector* and *ActedOnly*. (We explicitly show the variable quantifiers in the views as they will be needed to construct the first-order lineage formulas.) View ν_1, for example, expresses that directors are known for a movie category if they occur in the relation *BestDirector* together with a movie of that category. Likewise, view ν_2 expresses that actors are known for movies that won a best picture award, but only if they appear together in the *ActedOnly* relation. Fig. 1 also depicts a partially evaluated lineage formula for the query $\mathit{KnownFor}(X, \mathit{Crime})$ over the views and base tuples of our example, thus asking for directors or actors X who are known for Crime movies. As is shown by the figure, the lineage encodes a first-order formula that captures an intermediate step of processing the query via the views in a top-down fashion. Both ν_1 and ν_2 have only been partially resolved to their body literals (aka. "subgoals" in Datalog terminology), but the remaining part of the lineage is still unexplored. ⋄

A. Contributions

- To our knowledge, our approach is the first to consider *integrated data and confidence computations* for queries that do not permit safe query plans and hence do not allow for efficient extensional query evaluations [1], [2]. Thus, early pruning of low-confidence answer candidates may yield significantly reduced data computation and storage costs in PDBs with non-materialized views.
- We present a generic bounding approach for confidence computations over *first-order lineage* formulas. Our algorithm provides *lower* and *upper bounds* for the marginal probability of an individual query answer, or for an entire set of query answers if not all query variables are bound to constants yet. We show that both our lower and upper bounds *converge monotonically* to the final confidences of the query answers as we gradually expand these formulas.
- Our approach allows for plugging in *different schedulers* which aim to select the subgoal (represented by a first-order literal inside a lineage formula) that is most beneficial for top-k pruning at each query processing step.
- We extend our algorithm for the case when *sorted input lists* for extensional relations are available, and for adapting our top-k pruning techniques to *recursive rules*.
- We present an extensive experimental evaluation and comparison to existing top-k pruning strategies in probabilistic databases. In particular, we are the first to report an improved runtime of our top-k algorithm in a probabilistic database setting in comparison to full query evaluations in a corresponding deterministic database setting.

II. Computational Model

In this section, we introduce our data model, which follows the common possible-worlds semantics [15] over tuple-independent probabilistic databases, along with the operations we allow over this kind of uncertain data. Our computational model builds upon (and thus is consistent with) prior work on probabilistic databases [7], [16], [17], [18], and specifically upon the one considered in the context of *uncertain databases with lineage* [12], [13], which is known to be *closed* and *complete* under the common semantics of relational operations.

A. Probabilistic Database

We define a *tuple-independent probabilistic database with views* $\mathcal{DB} = (\mathcal{T}, \mathcal{V}, p)$ as a triplet consisting of a set of *base tuples* $\mathcal{T}$, a set of *views* $\mathcal{V}$, and a *probability measure* $p : \mathcal{T} \rightarrow (0, 1]$ which assigns a probability value $p(t)$ to each uncertain base tuple $t \in \mathcal{T}$.[1] As in a regular database, we assume the set of base tuples $\mathcal{T}$ to be partitioned into a set of

[1] Usually a PDB is defined as a probability distribution over possible instances of the database. In the case of a tuple-independent PDB, this distribution corresponds to the one defined by Equation (1).

extensional relations. The probability value $p(t)$ denotes the confidence in the existence of the tuple in the database, i.e., a higher value $p(t)$ denotes a higher confidence in t being valid. Uncertainty of base tuples is modeled by associating a Boolean random variable $\mathcal{X}_t$ with each base tuple $t \in \mathcal{T}$. The case when $\mathcal{X}_t = true$ denotes the probabilistic event that t is present in the probabilistic database. We assume globally unique identifiers for base tuples. For convenience of notation, and if it is clear from the context, we will use t to denote both the identifier and the random variable $\mathcal{X}_t$ associated with t.

Possible Worlds Semantics. A *possible world* $\mathcal{W} \subseteq \mathcal{T}$ is a subset of base tuples in $\mathcal{T}$. Since we assume independence among all Boolean random variables associated with tuples, the *probability* $P(\mathcal{W})$ of a possible world $\mathcal{W}$ is defined as follows.

$$P(\mathcal{W}) := \prod_{t \in \mathcal{W}} p(t) \prod_{t \notin \mathcal{W}} (1 - p(t)) \tag{1}$$

Intuitively, all tuples in $\mathcal{W}$ are valid (i.e., *true*) in the possible world $\mathcal{W}$, whereas all tuples in $\mathcal{T} \backslash W$ are *false* (i.e., they are not contained in $\mathcal{W}$). In the absence of further constraints restricting the set of possible worlds, each subset of base tuples $\mathcal{W} \in 2^{\mathcal{T}}$ forms a valid possible world. Hence, there are exponentially many such possible worlds.

B. Views

We represent a view $\nu \in \mathcal{V}$ as a rule in Datalog notation. Hence $\mathcal{V}$ together with the set of base tuples (aka. “facts”) $\mathcal{T}$ is also called a *Datalog program*. We will denote the deductive query processing steps applied for processing these rules as *deductive grounding*. Syntactically, a Datalog rule is a disjunctive clause with a positive head literal and a conjunction of both positive and negative literals in its body (see Fig. 1 for examples). The views’ head literals define a set of *intensional relations*. An intensional relation may be defined via multiple rules; however, no extensional relation may occur as the head literal of a rule. Variables occurring in the head literal are universally quantified, while variables occurring only in the body literals are existentially quantified (see, e.g., ν_1 and ν_2 in Fig. 1). Following common Datalog conventions, each variable that occurs in the head literal or in a negated body literal must also occur in at least one of the positive body literals. This form of *safe Datalog* programs ensures that grounding terminates, and the variables are properly bound to constants after grounding the rules. For the rest of the paper, we will use the terms *view* and *rule* interchangeably.

We remark that we do not focus on safe query plans [2], and hence we do not pose any further restrictions on the views’ shape. Also, we note that this class of safe, non-recursive Datalog programs with negation corresponds to the class of queries expressible in unrestricted relational algebra (however without grouping and aggregations) [19]. For any given instance of non-recursive Datalog rules, the data complexity is of polynomial time in the size of the base tuples [20].

C. Queries

We consider a *query* as a conjunction of first-order literals whose arguments are tuples of constants and free (aka. “distinguished”) variables, which we will refer to as the *query variables*. Again, every variable occurring in a negated literal must also occur in at least one of the non-negated literals. Tuples of constants, which become bound to tuples of query variables by the grounding procedure, yield the query answers.

D. Lineage

In contrast to base tuples, which are assumed to be independent, a derived tuple is completely defined via (and thus dependent of) the base tuples that were employed to derive that tuple. Thus, when completely grounded against the base tuples, we will refer to a derived tuple t directly via its propositional lineage formula ϕ_t.

As opposed to all probabilistic database approaches we are aware of (see, e.g., [1], [12], [14]), which consider lineage only in propositional form, we more generally allow lineage to be a well-formed formula Φ over a restricted class of first-order predicate logic. A *well-formed lineage formula* may incorporate the Boolean constants *true* and *false*, Boolean connectives ($\wedge$, $\vee$, $\neg$), Boolean (random) variables denoting tuples $t \in \mathcal{T}$, existential quantifiers ($\exists$), and first-order literals of the form $R^{\gamma}(\bar{X})$. Following common Datalog terminology, we refer to a first-order literal $R^{\gamma}(\bar{X})$ as a *subgoal*, where R denotes the relation name and $\bar{X}$ is a tuple consisting of both constants and variables. Subgoals represent yet unexplored (i.e., not yet grounded) parts of the lineage formula. We employ *adornments* in the form of a superscript γ to denote which variables of a subgoal are bound or free.

As opposed to propositional lineage, a first-order lineage formula is able to capture any intermediate step of a top-down grounding procedure. If at least one query variable in a first-order lineage formula is not yet bound to a constant, the lineage formula represents a (potentially empty) set of query answers. In the example in Fig. 1, the single query variable X is not yet bound, and hence the lineage formula captures all answers which can be obtained by binding X to constants.

E. Deductive Grounding & Lineage

We next provide an inductive definition of *lineage* which is obtained from grounding a subgoal $R^{\gamma}(\bar{X})$ over views $\mathcal{V}$ and uncertain base tuples $\mathcal{T}$. The definition is based on two rewriting rules which follow the general course of a top-down grounding procedure. We choose top-down grounding over bottom-up grounding [19] in order to be able to save data computations, i.e., to avoid touching base tuples of lower ranked answers whenever possible. In Section V, we provide a grounding algorithm, based on *SLD resolution* [19], which implements these two rewriting rules.

Rule (1) *(Disjunctive Lineage)* Let $R^{\gamma}(\bar{X})$ be a subgoal, and let $\bar{X}$ be a tuple of constants and variables not bound in γ. Then grounding $R^{\gamma}(\bar{X})$ over views $\mathcal{V}$ and base tuples $\mathcal{T}$ yields a disjunction over the lineages of base tuples or tuples derived from views that unify with $R^{\gamma}(\bar{X})$.

$$\Phi\left(R^{\gamma}(\bar{X})\right) := \begin{cases} \bigvee_{\nu}\left(\Phi(\nu)\right) & \bullet \text{ if } R \text{ is intensional} \\ & \quad \text{and } head(\nu) \text{ of } \nu \in \mathcal{V} \\ & \quad \text{unifies with } R^{\gamma}(\bar{X}) \\ \bigvee_{t} \mathcal{X}_t & \bullet \text{ if } R \text{ is extensional} \\ & \quad \text{and } t \in \mathcal{T} \text{ unifies} \\ & \quad \text{with } R^{\gamma}(\bar{X}) \\ false & \bullet \text{ else} \end{cases}$$

Rule (2) *(Conjunctive Lineage)* Let $R^{\gamma}(\bar{X})$ be a subgoal, and let $\bar{X}$ be a tuple of constants and variables not bound in γ. Further, let

$$\nu:\ \forall \bar{X}'\ R(\bar{X_0}) \coloneq \exists \bar{X}''\ L_1(\bar{X_1}), \ldots, L_n(\bar{X_n})$$

be a safe Datalog rule whose head literal $R(\bar{X_0})$ unifies with $R^{\gamma}(\bar{X})$, and let $\bar{X}_0, \ldots, \bar{X}_n$ be tuples of constants and existentially quantified variables. Then grounding $R^{\gamma}(\bar{X})$ against ν yields a conjunction over the lineages of literals $L_1, \ldots, L_n$ in the body of ν.

$$\Phi\left(\nu\right) := \exists \bar{X}'' \left(\bigwedge_{i=1,\ldots,n} \begin{cases} \Phi\left(R_i^{\gamma}(\bar{X}_i)\right) & \bullet \text{ if } L_i = R_i \\ \neg\left(\Phi\left(R_i^{\gamma}(\bar{X}_i)\right)\right) & \bullet \text{ if } L_i = \neg R_i \end{cases} \right)$$

Query Processing. For a subgoal $R^{\gamma}(\bar{X})$ over an extensional relation R, only Rule (1) applies. It replaces $R^{\gamma}(\bar{X})$ by either a disjunction of Boolean variables representing base tuples or by the constant *false*, if no such tuples exist (which corresponds to the common "negation-as-failure" semantics in Datalog [19]). If R is intensional, Rule (1) is utilized to create a disjunction over all rules whose head literal unifies with the subgoal. Then, the subsequent application of Rule (2) results in a conjunction of literals in each such rule's body, where existential quantifiers over the variables that occur in the rule's body are added to the lineage. In SLD resolution, this process is repeated by using the body literals of the rule as new subqueries in the subsequent grounding steps.

Creating Query Answers. If a tuple of arguments $\bar{X}$ of a subgoal $R^{\gamma}(\bar{X})$ becomes bound to one or more tuples of constants $\bar{C}_1, \ldots, \bar{C}_n$, we distinguish two cases. First, if $R^{\gamma}(\bar{X})$ relates to a top-level query literal, then each distinct tuple $\bar{C}_i$ corresponds to a new query answer and its lineage is copied correspondingly. Second, if $\bar{X}$ contains existentially quantified variables, then these can be eliminated through a standard quantifier elimination step [19]. In general, if the bindings to a variable X in Φ are $C_1, \ldots, C_n$, then we transform Φ into a disjunction of formulas $\Phi_{[X \rightarrow C_i]}$ as follows.

$$\exists X \Phi \equiv \Phi_{[X \rightarrow C_1]} \vee \cdots \vee \Phi_{[X \rightarrow C_n]} \tag{2}$$

In this case, no new answers are introduced, but the quantifier elimination results in a corresponding disjunction in the lineage formula that is currently being processed.

Complete Lineage. In a non-probabilistic Datalog setting, it is sufficient to find a single deductive proof for an answer to show that this answer exists. In contrast, for Datalog rules over probabilistic data, *all* such proofs over the given rules and base tuples are required to correctly capture all the possible worlds (and only those) for which a query answer exists. SLD resolution yields this "all-proofs" semantics [21].

Example 2: For the query $KnownFor(X, Crime)$ over the base tuples and views of Fig. 1, we observe that the head literals of both ν_1 and ν_2 unify with this query literal. Applying one step of SLD resolution along with the two lineage rewriting rules, Rule (1) and (2), to the query literal yields the following lineage formula (also depicted in Fig. 1).

$$\begin{gathered} \exists Z \begin{pmatrix} BestDirector(X,Z) \\ \wedge\ Category(Z, Crime) \end{pmatrix} \\ \bigvee \\ \exists Z \begin{pmatrix} WonAward(Z, BestPicture) \\ \wedge\ ActedOnly(X,Z) \\ \wedge\ Category(Z, Crime) \end{pmatrix} \end{gathered}$$

In the next two SLD steps, we resolve the two remaining intensional subgoals $BestDirector(X, Z)$ and $ActedOnly(X, Z)$ via views ν_3 and ν_4, respectively.

$$\begin{gathered} \exists Z \begin{pmatrix} \begin{pmatrix} Directed(X,Z) \\ \wedge\ WonAward(Z, BestDirector) \end{pmatrix} \\ \wedge\ Category(Z, Crime) \end{pmatrix} \\ \bigvee \\ \exists Z \begin{pmatrix} WonAward(Z, BestPicture) \\ \wedge\ \left(Acted(X,Z) \wedge \neg Directed(X,Z)\right) \\ \wedge\ Category(Z, Crime) \end{pmatrix} \end{gathered}$$

Finally, by applying the first rewriting rule, Rule (1), to the remaining extensional subgoals, we obtain the two possible query answers $KnownFor(Coppola, Crime)$ and $KnownFor(Tarantino, Crime)$ with lineages $(t_2 \wedge t_8 \wedge t_{12})$ and $(t_{10} \wedge (t_6 \wedge \neg t_3) \wedge t_{13})$, respectively. $\diamond$

F. Confidence Computations

For a propositional lineage formula ϕ, let $\mathcal{M}(\phi)$ be the set of possible worlds (aka. "models") satisfying ϕ. Then, the *marginal probability* $P(\phi)$ of a derived tuple (represented by its propositional lineage formula ϕ) is defined as the sum of the probabilities of all the possible worlds for which ϕ evaluates to *true*.

$$P(\phi) := \sum_{\mathcal{W} \in \mathcal{M}(\phi)} P(\mathcal{W}) \tag{3}$$

We note that the above sum may range over exponentially many terms because there are exponentially many possible worlds. In fact, computing $P(\phi)$ is known to be $\#\mathcal{P}$-hard for general propositional formulas [2], [3].

Alternatively, to avoid computing the sum of Equation (3), we can compute marginals by incrementally decomposing the propositional lineage formulas into variable-disjoint subformulas [8], [22]. Generally, for two propositional formulas ϕ, ψ over disjoint sets of independent random variables, the following relationships hold:

$$P(\phi \wedge \psi) := P(\phi) \cdot P(\psi) \tag{4}$$

$$P(\phi \vee \psi) := 1 - (1 - P(\phi)) \cdot (1 - P(\psi)) \tag{5}$$

$$P(\neg \phi) := 1 - P(\phi) \tag{6}$$

If the above principles are not directly applicable to ϕ and ψ due to a shared variable t, this variable can be eliminated by a *Shannon expansion*. This is based on the equivalence

$$\phi \equiv (t \wedge \phi_{[t \to true]}) \vee (\neg t \wedge \phi_{[t \to false]})$$

where $\phi_{[t \to true]}$ denotes the restriction of ϕ to the case when t is *true*, i.e., all occurrences of t in $\phi_{[t \to true]}$ are substituted by the constant *true*. Then, it holds that:

$$P(\phi) = P(t)\ P(\phi_{[t \to true]}) + (1 - P(t))\ P(\phi_{[t \to false]}) \quad (7)$$

Repeated Shannon expansions can increase the size of a formula exponentially. This issue can be addressed to some extent by incremental decompositions as shown in [22].

III. Confidence Bounds

In this section, we develop lower and upper bounds for the marginal probability of any query answer that can be obtained from grounding a first-order lineage formula. We will proceed by constructing two propositional lineage formulas ϕ_{low} and ϕ_{up} from a given first-order lineage formula Φ. Then, the confidences of ϕ_{low} and ϕ_{up} will serve as lower and upper bounds on the confidences of all query answers captured by Φ. More formally, if $\phi_1, \ldots, \phi_n$ represent all query answers we would obtain by fully grounding Φ, then it holds that:

$$\forall i \in \{1, \ldots, n\} : \ P(\phi_{low}) \leq P(\phi_i) \leq P(\phi_{up})$$

Building upon results of [8], [22], [23], we next develop two theorems, which (1) provide a mechanism for obtaining lower and upper bounds for formulas with first-order literals, and which (2) guarantee that these bounds converge monotonically to the marginal probabilities $P(\phi_i)$ of each query answer ϕ_i as we continue to ground Φ.

A. Bounds for Propositional Lineage

As a first step, we relate the confidence of two propositional lineage formulas ϕ and ψ via their sets of models $\mathcal{M}(\phi)$ and $\mathcal{M}(\psi)$, i.e., the sets of possible worlds over which ϕ and ψ evaluate to *true*, respectively.

Proposition 1: Following [22], for two propositional lineage formulas ϕ and ψ, it holds that:

$$\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi) \quad \Rightarrow \quad P(\phi) \leq P(\psi)$$

That is, $\mathcal{M}(\phi)$ includes all possible worlds for which ϕ evaluates to *true*. Since we assume $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$, the same worlds satisfy ψ as well. However, there might be more worlds fulfilling ψ but not ϕ. This might yield more terms over which the sum of Equation (3) ranges, and thus we obtain $P(\phi) \leq P(\psi)$.

Example 3: Consider the two propositional formulas $\phi \equiv t_1$ and $\psi \equiv t_1 \vee t_2$. From $\mathcal{M}(t_1) \subset \mathcal{M}(t_1 \vee t_2)$ it follows that $P(t_1) < P(t_1 \vee t_2)$, which we can easily verify using Equation (3). $\diamond$

Conjunctive Clauses. To turn Proposition 1 into upper and lower bounds, we proceed by considering *conjunctive clauses* in the form of conjunctions of propositional literals, where $Literals(\phi)$ denotes the set of literals contained in a clause.

Proposition 2: Let ϕ, ψ be two propositional, conjunctive clauses. It holds, that $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$ if and only if $Literals(\phi) \supseteq Literals(\psi)$ *[22].*

The above statement expresses that adding literals to a conjunction ϕ removes satisfying worlds from $\mathcal{M}(\phi)$.

Example 4: For the two clauses $t_1 \wedge t_2$ and t_1, it holds that $Literals(t_1 \wedge t_2) \supseteq Literals(t_1)$ and thus $\mathcal{M}(t_1 \wedge t_2) \subseteq \mathcal{M}(t_1)$. $\diamond$

Disjunctive Normal Form. Moreover, we say that a propositional formula ϕ is in *disjunctive normal form* (DNF), if it is a disjunction of conjunctive clauses.

Lemma 1: For two propositional DNF formulas ϕ and ψ, it holds that

$$\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi) \quad \Leftrightarrow \quad \forall \phi' \in \phi\ \exists \psi' \in \psi : \mathcal{M}(\phi') \subseteq \mathcal{M}(\psi')$$

where ϕ' and ψ' are conjunctive clauses *[22], [23].*

The lemma establishes a relationship between two formulas in DNF. If we can map all clauses ϕ' of a formula ϕ to a clause ψ' of ψ with more satisfying worlds, i.e., $\mathcal{M}(\phi') \subseteq \mathcal{M}(\psi')$, then ψ has more satisfying worlds than ϕ. This mapping of clauses is established via Proposition 2.

Example 5: For the propositional DNF formula $\phi \equiv (t_1 \wedge t_2) \vee (t_1 \wedge t_3) \vee t_4$, we can map each clause in ϕ to a clause in $\psi \equiv t_1 \vee t_4$. Hence, ψ has more models than ϕ, i.e., $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$. $\diamond$

Thus, Lemma 1 together with Proposition 1 enables us to compare the marginal probabilities of propositional formulas in DNF based on their clause structure.

Any propositional formula can equivalently be transformed into DNF by iteratively applying De Morgan's law and thereafter the distributive law.

Observation 1: If a variable t occurs exactly once in a propositional formula ϕ, then all occurrences of t in the DNF of ϕ have the same sign.

The reason is that the sign of a variable t changes only by using De Morgan's law. However, when applying De Morgan's law, no variables are duplicated. When utilizing the distributive law, variables are duplicated but preserve their signs.

B. Bounds for First-Order Lineage

Analogously to the DNF for propositional formulas, any first-order formula can equivalently be transformed into prenex normal form by pulling all quantifiers in front of the formula. The remaining formula can again be transformed into DNF, which is then called prenex DNF (PDNF). For our following constructions on first-order formulas, we will assume the first-order formulas to be given in PDNF. In general, such a normalization may lead to an exponential increase of the size of the formula. However, this construction is employed for theoretical considerations only, and never actually needs to be performed by the algorithms described in Section V.

Assume we are given a first-order lineage formula Φ_R which is in propositional form except for one subgoal $R^\gamma(\bar{X})$. We also require the grounding of $R^\gamma(\bar{X})$ (see Section II-D) to yield only propositional terms, i.e., Boolean variables referring to base tuples. Hence, we refer to $\phi_1, \ldots, \phi_n$ as the propositional formulas, which we obtain by grounding $R^\gamma(\bar{X})$ in Φ_R.

Following ideas for propositional lineage formulas from [24], Theorem 1 provides bounds on each $P(\phi_i)$ by means of Φ_R.

Theorem 1: Given a first-order lineage formula Φ_R, which is in propositional form except for one subgoal $R^\gamma(\bar{X})$, and propositional lineage formulas $\phi_1, \ldots, \phi_n$, which are obtained from Φ_R by grounding $R^\gamma(\bar{X})$.
We construct ϕ_{up} by substituting $R^\gamma(\bar{X})$ with

- *true* if $R^\gamma(\bar{X})$ occurs positive in the PDNF of Φ_R, or
- *false* if $R^\gamma(\bar{X})$ occurs negated in the PDNF of Φ_R.

We construct ϕ_{low} by substituting $R^\gamma(\bar{X})$ with

- *false* if $R^\gamma(\bar{X})$ occurs positive in the PDNF of Φ_R, or
- *true* if $R^\gamma(\bar{X})$ occurs negated in the PDNF of Φ_R.

Then it holds that:

$$\forall i \in \{1, \ldots, n\} : \ P(\phi_{low}) \leq P(\phi_i) \leq P(\phi_{up})$$

Proof: Choose an arbitrary but fixed $i \in \{1, \ldots, n\}$. W.l.o.g., we assume Φ_R and ϕ_i to be in PDNF. The PDNF of Φ_R may consist of one or more clauses that contain $R^\gamma(\bar{X})$, which are either of the form $(\psi \wedge R^\gamma(\bar{X}))$ or $(\psi \wedge \neg R^\gamma(\bar{X}))$. For each of these clauses, ϕ_i may contain a number (due to Equation (2)) of clauses of the form $(\psi \wedge \varphi)$. Here, the literals φ correspond to groundings of $R^\gamma(\bar{X})$.

When substituting $R^\gamma(\bar{X})$ by *true* or *false* as stated in Theorem 1, Φ_R's clauses turn into $(\psi \wedge \mathit{true})$ and $(\psi \wedge \mathit{false})$ for the upper and lower bounds, respectively. Subsequently, considering the upper bound, we employ Proposition 2 which yields $\mathcal{M}(\psi \wedge \varphi) \subseteq \mathcal{M}(\psi \wedge \mathit{true})$, since $\mathit{Literals}(\psi \wedge \varphi) \supseteq \mathit{Literals}(\psi \wedge \mathit{true})$. Next, from Lemma 1 it follows that $\mathcal{M}(\phi_i) \subseteq \mathcal{M}(\phi_{up})$. This matches exactly the precondition of Proposition 1, from which we obtain $P(\phi_i) \leq P(\phi_{up})$. The case for the lower bound $P(\phi_{low})$ follows analogously. ■

Example 6: For the first-order lineage formula

$$\Phi_R \equiv t_{11} \wedge \exists X \, \mathit{ActedIn}(\mathit{Brando}, X)$$

the upper bound is given by $P(t_{11} \wedge \mathit{true}) = P(t_{11}) = p(t_{11})$ and the lower bound is $P(t_{11} \wedge \mathit{false}) = P(\mathit{false}) = 0$. Thus, for any set of tuples $t_1, \ldots, t_n$ matching $\exists X \ \mathit{ActedIn}(\mathit{Brando}, X)$, we have $\phi \equiv (t_{11} \wedge (t_1 \vee \cdots \vee t_n))$ with $0 \leq P(t_{11} \wedge (t_1 \vee \cdots \vee t_n)) \leq P(t_{11})$. $\diamond$

Since $R^\gamma(\bar{X})$ has exactly one occurrence in Φ_R, all occurrences of $R^\gamma(\bar{X})$ in the PDNF of Φ_R have the same sign (see Observation 1). Therefore, we replace all occurrences of $R^\gamma(\bar{X})$ in the PDNF of Φ_R by either *true* or *false*.

More generally, for a general first-order lineage formula Φ, which contains multiple distinct subgoals, we can apply the substitution provided by Theorem 1 multiple times to obtain these lower and upper bounds. That is, we replace every occurrence of a subgoal $R^\gamma(\bar{X})$ in Φ by one application of Theorem 1's substitution to obtain ϕ_{low} and ϕ_{up} of Φ.

Convergence of Bounds. Our last step is to show that, for a fixed query answer ϕ (see Section II-E), the confidence bounds converge monotonically to the marginal probability of the propositional lineage formula $P(\phi)$ with each SLD step. The argument follows from the execution of the grounding procedure, but backwards. In the following, let Φ_1 denote the original query, and let Φ_n correspond to the lineage formula before the last grounding step, from which we obtain the final propositional formula ϕ.

Theorem 2: Let $\Phi_1, \ldots, \Phi_n$ denote a series of first-order formulas obtained from iteratively grounding a conjunctive query via SLD resolution. Then, rewriting each Φ_i to $\phi_{i,low}$ and $\phi_{i,up}$ according to Theorem 1 creates a monotonic series of lower and upper bounds $P(\phi_{i,low})$, $P(\phi_{i,up})$ for the marginal probability $P(\phi)$. That is:

$$\begin{aligned} 0 \leq P(\phi_{1,low}) \leq \cdots \leq P(\phi_{n,low}) \leq P(\phi) \\ \leq P(\phi_{n,up}) \leq \cdots \leq P(\phi_{1,up}) \leq 1 \end{aligned}$$

Proof: We prove the theorem by induction, where i denotes the number of grounding steps taken.
Basis $i = n$: $P(\phi_{n,low}) \leq P(\phi) \leq P(\phi_{n,up})$ is covered by one application of Theorem 1. That is, we have exactly one occurrence of a subgoal $R^\gamma(\bar{X})$ in Φ_n, which we replace with either *true* or *false* to obtain $\phi_{n,low}$ and $\phi_{n,up}$, respectively, such that $P(\phi_{n,low}) \leq P(\phi)$ and $P(\phi_{n,up}) \geq P(\phi)$.
Step $i \rightarrow i-1$: By the hypothesis, we are given a formula Φ_i with bounds characterized by $P(\phi_{i,up})$ and $P(\phi_{i,low})$.

Let us consider the grounding step which led to Φ_i. In Φ_{i-1} a subgoal $R^\gamma(\bar{X})$ must have been processed from which we obtained Φ_i. Let $\Phi_{i-1} \equiv \Psi \ R^\gamma(\bar{X}) \ \Psi'$, where Ψ, Ψ' are a prefix and suffix of Φ_{i-1}. One step of grounding $R^\gamma(\bar{X})$ via SLD resolution thus leads to the formula $\Phi_i \equiv \Psi \ \Phi'_i \ \Psi'$, where Φ'_i is a first-order formula that consists of one or more subgoals or ground terms (including the constants *true* and *false*). If we replace every occurrence of $R^\gamma(\bar{X})$ in the conjunctive clauses of the PDNF of Φ_{i-1} by Φ'_i, we obtain a formula that is equivalent to Φ_i (but which is not necessarily in PDNF), and whose clauses contain more terms than the clauses in Φ_{i-1} and hence are more specific. From this, it follows that applying Theorem 1 to all subgoals in the actual PDNF's of Φ_i and Φ_{i-1} yields propositional formulas $\phi_{i,up}$ and $\phi_{i-1,up}$, such that $\mathcal{M}(\phi_{i,up}) \subseteq \mathcal{M}(\phi_{i-1,up})$. That is, $P(\phi_{i,up}) \leq P(\phi_{i-1,up})$. Again, the case for the lower bounds $P(\phi_{i,low})$ follows analogously. ■

IV. Subgoal Scheduling

We now present our scheduling techniques for determining the benefit of exploring a particular subgoal. A major difference of Datalog to a traditional DB setting is the lack of a static query plan. Instead, we aim to *dynamically and adaptively* determine the best join order among the literals in a rule's body at each grounding step. As in a deterministic setting, we aim to ground subgoals with a low selectivity first. Given our probabilistic setting, we moreover prioritize those subgoals that also have a high impact on the confidence of the answers.

A. Selectivity Estimation

Selectivity estimation aims at computing how many answers are expected from the database when a subgoal is expanded. We employ a simple probabilistic model defined over both the view structure and the extensional relations, with independence assumptions for joins and unions. We will express this by a function $S : \Phi \to [0,1]$ that reflects the likelihood of obtaining results when Φ is grounded.

We recursively define the *selectivity* $S(\Phi)$ of a lineage formula Φ involving only extensional subgoals of the form $R^{\gamma}(\bar{X})$ with binding pattern γ and Boolean connectives $\wedge$, $\vee$ and $\neg$ as follows:

$$\begin{array}{lll} S(R^{\gamma}(\bar{X})) & := & s_R^{\gamma} \text{ if } R \text{ is extensional} \\ S(\neg\Phi) & := & 1 - S(\Phi) \\ S(\bigwedge_{i=1}^{n} \Phi_i) & := & \prod_{i=1}^{n} S(\Phi_i) \\ S(\bigvee_{i=1}^{n} \Phi_i) & := & 1 - \prod_{i=1}^{n} (1 - S(\Phi_i)) \end{array}$$

Here, s_R^{γ} denotes the selectivity of a subgoal $R^{\gamma}(\bar{X})$ over an extensional relation R given the binding pattern γ. To avoid computing s_R^{γ} for all constants that could possibly occur in γ, we approximate it as follows. Given an extensional relation R, we take the average amount of tuples in R that match the free variables of γ grouped by the constants in this set of tuples, and divide this value by the size of the database $|\mathcal{T}|$.

Example 7: Let us again consider the two intensional subgoals $\mathit{BestDirector}(X,Z)$ and $\mathit{ActedOnly}(X,Z)$ (with binding pattern $\gamma_1 = \{Y{=}\,\mathit{Crime}\}$), which we obtain from grounding the query $\mathit{KnownFor}(X, \mathit{Crime})$ over views ν_1–ν_4 in Fig. 1. After resolving the subgoals to extensional relations via the views, selectivity estimation proceeds as follows. For the selectivity of $\mathit{BestDirector}(X,Z)$, we obtain

$$\begin{array}{l} S\left(\mathit{BestDirector}^{\gamma_1}(X,Z)\right) \\ = S\left(\mathit{Directed}^{\gamma_2}(X,Z) \wedge \mathit{WonAward}^{\gamma_2}(Z, \mathit{BestDirector})\right) \\ = \frac{3}{14} \cdot \frac{4}{3 \cdot 14} \approx 0.020 \end{array}$$

where $\gamma_2 = \{Y{=}\,\mathit{Crime}\}$ and $\frac{3}{14}$ results from the fact that *Directed* returns 3 tuples over a total of 14 in the database when none of the arguments are bound. Instead, *WonAward* on average returns $(1+1+2)/3 = 4/3$ tuples (again over a total of 14 in the database) when the second argument is bound. For the selectivity of $\mathit{ActedOnly}(X,Z)$, we obtain

$$\begin{array}{l} S\left(\mathit{ActedOnly}^{\gamma_1}(X,Z)\right) \\ = S\left(\mathit{Acted}^{\gamma_2}(X,Z) \wedge \neg \mathit{Directed}^{\gamma_2}(X,Z)\right) \\ = \frac{3}{14} \cdot (1 - \frac{3}{14}) \approx 0.168 \end{array}$$

where again $\gamma_2 = \{Y{=}\,\mathit{Crime}\}$ and $\frac{3}{14}$ results from the fact that both *Acted* and *Directed* return 3 tuples over a total of 14 in the database when none of their arguments are bound. Notice that the algorithm takes into account the propagation of binding patterns from γ_1 to γ_2 at the subsequent grounding step via SLD resolution (see also Algorithm 2). $\diamond$

B. Impact of Subgoals

In the next step, we aim to quantify the impact of the confidence $p(t)$ of a Boolean variable t on the marginal probability $P(\phi)$ of a propositional formula ϕ in which t occurs. Later, the scheduler will exploit this to choose the subgoal with the highest impact on the confidence bounds of ϕ. Following results from [5], [25], this impact measure can be captured by the following derivative.

$$\frac{\partial P(\phi)}{\partial p(t)} = \frac{P(\phi_{[t \to true]}) - P(\phi_{[t \to false]})}{1 - 0}$$

Lemma 2: For a propositional formula ϕ, if we fix the confidences of all variables except t, it holds that

$$P(\Phi) = c\, p(t) + c'$$

where c and c' are two constants that are independent of $p(t)$.

Proof: One step of Shannon Expansion on t results in:

$$\begin{array}{l} \phi \equiv (t \wedge \phi_{[t \to true]}) \vee (\neg t \wedge \phi_{[t \to false]}) \\ \Rightarrow \\ P(\phi) = p(t)\ P(\phi_{[t \to true]}) + (1 - p(t))\ P(\phi_{[t \to false]}) \\ \quad = p(t)\ \underbrace{(P(\phi_{[t \to true]}) - P(\phi_{[t \to false]}))}_{c} + \underbrace{P(\phi_{[t \to false]})}_{c'} \end{array}$$

■

Thus, to compute the above derivative, it suffices to compute c. A general first-order lineage formula Φ, however, may contain subgoals which makes the above sensitivity analysis not directly applicable to Φ. Again, by substituting all subgoals in Φ according to Theorem 1, we can quantify the impact of a subgoal $R^{\gamma}(\bar{X})$ on both the upper and lower bounds $P(\phi_{low})$, $P(\phi_{up})$ in the corresponding propositional formulas ϕ_{low}, ϕ_{up} of Φ. That is, to quantify the impact of $R^{\gamma}(\bar{X})$ on the upper bound, we substitute all other subgoals to obtain ϕ_{up} and then compute c by substituting $R^{\gamma}(\bar{X})$ once by the constant *true* and once by *false*.

Example 8: Consider the first-order lineage formula

$$\begin{array}{c} \left(\mathit{BestDirector}(\mathit{Coppola}, \mathit{Godfather}) \wedge t_{12}\right) \\ \vee \begin{pmatrix} t_9 \wedge \mathit{ActedOnly}(\mathit{Coppola}, \mathit{Godfather}) \\ \wedge \mathit{Category}(\mathit{Godfather}, \mathit{Crime}) \end{pmatrix} \end{array}$$

with the subgoals $\mathit{BestDirector}(\mathit{Coppola}, \mathit{Godfather})$, $\mathit{ActedOnly}(\mathit{Coppola}, \mathit{Godfather})$, and $\mathit{Category}(\mathit{Godfather}, \mathit{Crime})$ (which corresponds to the first query answer of Example 2). The impact of $\mathit{BestDirector}(\mathit{Coppola}, \mathit{Godfather})$ on the formula's upper bound is calculated as:

$$\begin{array}{l} 1 - (1 - P(true) \cdot p(t_{12}))(1 - p(t_9) \cdot P(true) \cdot P(true)) - \\ \quad (1 - (1 - P(false) \cdot p(t_{12}))(1 - p(t_9) \cdot P(true) \cdot P(true))) \\ = 1 - (1 - 1 \cdot 0.5)(1 - 0.4 \cdot 1 \cdot 1) - \\ \quad (1 - (1 - 0 \cdot 0.5)(1 - 0.4 \cdot 1 \cdot 1)) \\ = 1 - 0.3 - (1 - 0.6) = 0.3 \ \diamond \end{array}$$

C. Benefit-oriented Subgoal Scheduling

We now define the combined *benefit* of scheduling a subgoal R^{γ} occurring in a lineage formula Φ as

$$\mathit{ben}(\Phi, R^{\gamma}) := \frac{|\mathit{im}_{up}(\Phi, R^{\gamma})| + |\mathit{im}_{low}(\Phi, R^{\gamma})|}{1 + S(R^{\gamma})}$$

where $\mathit{im}_{up}(\Phi, R^{\gamma})$ and $\mathit{im}_{low}(\Phi, R^{\gamma})$ denote the impact of R^{γ} on the upper and lower bound lineage formulas of Φ we obtain from Theorem 1. As an additional scheduling rule, we always prefer intensional over extensional subgoals, if we have the choice among both types of subgoals, and we perform one SLD step at a time to resolve the intensional subgoals. We remark that, albeit this form of scheduling employs a fairly simple form of selectivity estimation for joins and unions, it can be applied also to recursive Datalog rules, which is an extension of our data model we consider in Section VI-B.

V. Top-k Algorithm

Our top-k algorithm primarily operates on the lineage formulas of answer candidates. Specifically, subgoals from all answer candidates are kept in a priority queue Q ordered according to ben, the benefit function described in Section IV. Moreover, we maintain two disjoint sets of answer candidates A_{top} and A_{cand}. Following the seminal line of threshold algorithms [9], A_{top} comprises the current top-k answers with respect to the lower confidence bounds, while A_{cand} consists of all remaining answer candidates whose upper confidence bounds are still higher than the worst lower bound of any of the top-k answers. As an additional constraint, the top-k set A_{top} consists only of query answers whose lower bound is greater than 0. This coincides with those answers, for which all query variables have already been bound to constants by the grounding procedure, i.e., those for which we have at least one proof, but not necessarily all proofs yet. The candidate set, on the other hand, may also hold answer candidates with a lower confidence bound of 0, i.e., also those for which the query variables are not yet bound to constants.

```
Algorithm 1 Top-$k$($\mathcal{V}$, $\mathcal{T}$, $\Phi_q$, $k$)
Input: Views $\mathcal{V}$, uncertain tuples $\mathcal{T}$, an intensional query $\Phi_q$, and
    an integer value $k$
Output: Top-$k$ answers $A_{top}$ for $\Phi_q$ according to their lower confidence bounds
 1: Initialize a global priority queue $Q$ with subgoals from $\Phi_q$
 2: $A_{top} := \emptyset$                                  ▷ Current top-$k$ answers
 3: $A_{cand} := \{\Phi_q\}$                                ▷ Answer candidates
 4: while $A_{cand} \neq \emptyset$ do
 5:     min-$k$ := $\min_{\Phi_i \in A_{top}} \{P(\phi_{i,low}), 0\}$          ▷ Thm. 1
 6:     max-cand := $\max_{\Phi_i \in A_{cand}} P(\phi_{i,up})$             ▷ Thm. 1
 7:     if min-$k$ > max-cand then
 8:         break
 9:     $\left(\Phi_{best}, R^{\gamma}_{best}(\bar{X})\right) :=$
            $\arg\max_{\left(\Phi_i \in A_{top} \cup A_{cand}, R_i(\bar{X}) \in Q\right)} ben(\Phi_i, R_i(\bar{X}))$
                                                            ▷ Eqn. IV-C
10:     $\Phi$ := SLD($\mathcal{V}$, $\mathcal{T}$, $\Phi_{best}$, $R^{\gamma}_{best}(\bar{X})$)          ▷ Alg. 2
11:     Update $A_{top}$, $A_{cand}$, and $Q$ using $\Phi$
12: return $A_{top}$
```

A. Top-k with Dynamic Subgoal Scheduling

At each processing step, the scheduler chooses the currently best subgoal $R^{\gamma}_{best}(\bar{X})$ from the subgoal queue Q (Line 9), and, by using Algorithm 2, we expand the lineage formula of this subgoal by performing a single SLD step over both $\mathcal{V}$ and $\mathcal{T}$ (Line 10) as described in Section II-E. Then, we update A_{top}, and A_{cand} (Line 11) due to the following. First, expanding $R^{\gamma}_{best}(\bar{X})$ can change the confidence bounds of the answer. Second, if there are no matches to $R^{\gamma}_{best}(\bar{X})$, neither in $\mathcal{T}$ nor in $\mathcal{V}$, the answer candidates corresponding to $R^{\gamma}_{best}(\bar{X})$ may be deleted (and hence their lineage evaluates to *false*). And third, if a query variable was bound to more than one constant, one or more new top-k answer candidates are created.

We iteratively update Q (Line 11) to keep the subgoals it contains consistent with A_{top} and A_{cand}. First, all subgoals occurring in deleted answer candidates are dropped from Q. Next, we add all newly created subgoals (due to new answers or the grounding of rules) to Q. Last, the impact of all subgoals appearing in the same lineage formula as $R^{\gamma}_{best}(\bar{X})$ might have changed (see Section IV-B), and hence their priority in Q is updated. Algorithm 1 terminates when the threshold-based breaking condition (Line 7) of the algorithm holds, or when the candidate set A_{cand} runs out of valid answer candidates.

B. SLD Resolution with Lineage Tracing

Algorithm 2 covers a single SLD step and is called as a subroutine of Algorithm 1. During each SLD step, a subgoal $R^{\gamma}(\bar{X})$ is replaced by new subgoals obtained from grounding the rules that define $R^{\gamma}(\bar{X})$, such that an updated version of all answers' lineages that share $R^{\gamma}(\bar{X})$ is returned. The algorithm corresponds to Rules (1) and (2) introduced in Section II-E.

```
Algorithm 2 SLD($\mathcal{V}, \mathcal{T}, \Phi, R^{\gamma}(\bar{X})$)
Input: Views $\mathcal{V}$, uncertain tuples $\mathcal{T}$, a first-order lineage formula $\Phi$,
    a subgoal $R^{\gamma}(\bar{X})$ contained in $\Phi$
Output: Updated lineage formula $\Phi$
 1: if $R$ extensional then
 2:     $M := \{(t, \gamma_u) \mid t$ and $\gamma$ unify to $\gamma_u\}$
 3: else
 4:     $M := \left\{(\nu, \gamma_u) \;\middle|\; \begin{array}{c} \nu = R'^{\gamma'}(\bar{X}) \text{ :– } body \in \mathcal{V}, R = R', \\ \gamma \text{ and } \gamma' \text{ unify to } \gamma_u \end{array}\right\}$
 5: if $M = \emptyset$ then
 6:     Replace $R^{\gamma}(\bar{X})$ in $\Phi$ by false                     ▷ Rule (1)
 7:     return $\Phi$
 8: for $\gamma^*_u \in bindings(M)$ do
 9:     if $\gamma^*_u$ binds new variables in $\gamma$ then
10:         $\Phi$ := expand $\Phi$ utilizing Eqn. (2)
11:         $L := R^{\gamma^*_u}(\bar{X})$                        ▷ Created in previous step
12:     else
13:         $L := R^{\gamma}(\bar{X})$
14:     if $R$ is extensional then
15:         Replace $L$ by $\bigvee_{(t,\gamma_u) \in M, \gamma_u = \gamma^*_u} \mathcal{X}_t$                 ▷ Rule (1)
16:     else
17:         $B := \{body \mid (R'^{\gamma'}(\bar{X}) \text{:– } body, \gamma_u) \in M, \gamma_u = \gamma^*_u\}$
18:         for $body \in B$ do
19:             Propagate $\gamma^*_u$ to bindings in body's literals
20:         Replace $L$ in $\Phi$ by $\bigvee_{body \in B} body$             ▷ Rules (1),(2)
21: return $\Phi$
```

If R is extensional, we collect all matching tuples in M (Line 2). Otherwise, we gather all rules whose relation R' coincides with R and whose bindings γ' unify with γ (Line 4). Thus, the set M holds a pair consisting of both the rule and the unified bindings γ_u. If there are no matching rules or tuples, that is $M = \emptyset$, we replace the subgoal by *false* and return the altered Φ (Line 7). The loop in Line 8 ranges over all different bindings γ^*_u obtained from unifying the subgoal's bindings γ with the bindings γ' occurring in the head of a matching rule. If γ^*_u binds more variables than γ, then in Line 10 we instantiate the quantifiers that hold the newly bound variables according to Equation 2. Afterwards, the copy $R^{\gamma^*_u}(\bar{X})$ of $R^{\gamma}(\bar{X})$ is saved in L (Line 11). If $R^{\gamma^*_u}$ matches tuples from $\mathcal{T}$, we replace L in Φ by a disjunction of variables $\mathcal{X}_t$ in Line 15. Otherwise L is substituted by a disjunction over the bodies of all rules with head $R^{\gamma_u^*}$ (Line 20). For an illustration of the algorithm, we refer the reader to Example 2.

C. Final Result Ranking

Many applications that employ top-k queries require a complete ranking of the top-k answers. When Alg. 1 terminates, the marginal probabilities of the top-k answers may however not be known exactly—but only the bounds thereof. Similarly to the strategies in [6], we can tackle this either by iteratively running top-1,..., top-k queries, where an inspection of the k answer sets yields the desired ranking, or by continuing the grounding and decomposition steps until the confidence bounds of the top-k answers do not overlap anymore.

VI. Extensions

A. Sorted Input Relations

A powerful technique in top-k approaches for extensional data [9], [10], [26] is to store each relation in decreasing order of local ranks and to use the rank at the current scan position as an upper bound for the ranks of all remaining tuples. Along with a monotonic score aggregation function, this allows for the computation of monotonically decreasing upper bounds of answer candidates. In our setting, we rank query answers by their marginal probabilities which generally does not resolve to such a monotonic form of score aggregation. However, an extensional relation R may still contain a large amount of tuples that unify with a subgoal $R^\gamma(\bar{X})$. For example, for the top-2 answers of $Directed(X, Y)$ in Fig. 1, we could return only t_1 and t_2 if $Directed$ is sorted in decreasing order of $p(t_i)$. In Theorem 1, we replaced a subgoal $R^\gamma(\bar{X})$ by *true* (or *false* if the subgoal is negated) to obtain an upper bound for the lineage formula containing $R^\gamma(\bar{X})$. This corresponds to using 1 as a conservative upper bound for $R^\gamma(\bar{X})$, since $P(true) = 1$. According to the following observation, we can lower this upper bound for a subgoal over an extensional relation by exploiting this decreasing order of local ranks.

Observation 2: Let R be an extensional relation with tuples sorted in decreasing order of $p(t_i)$, let $R^\gamma(\bar{X})$ be a subgoal over R, and let $t_1, \dots, t_m$ be tuples in R that unify with $R^\gamma(\bar{X})$. Then, when grounding $R^\gamma(\bar{X})$, we can set the upper confidence bound of each tuple $t_j \in R$, with $i < j \leq m$, to $\min\{p(t_1), \dots, p(t_i)\}$, if and only if all unbound variables of $R^\gamma(\bar{X})$ are query variables and there exist no data duplicates in R with respect to $R^\gamma(\bar{X})$.

The key for this observation is that binding a query variable yields a new query answer, while binding existentially quantified variables introduced by a rule results in a disjunction in the lineage formula due to a quantifier elimination. This disjunction may result in a higher confidence than that of the individual input tuples due to Equation (5). For example, if two independent tuples t_1, t_2 with a confidence of 0.5 each match a single subgoal with non-query variables, then we obtain $1 - (1 - 0.5) \cdot (1 - 0.5) = 0.75 > 0.5$. Thus, using an upper bound of 0.5 would be incorrect. Also notice that this strategy assumes R not to contain any data duplicates with regard to $R^\gamma(\bar{X})$. If necessary, we can enforce this by applying an independent-project operation [1] to the relation as a preprocessing step (for all required projections of R).

B. Recursive Rules

In this subsection, we develop an algorithmic extension for handling rules with recursively defined intensional relations. To ensure a safe semantics for the deductive grounding steps, we require the set of recursive rules $\mathcal{V}$ to be *stratifiable* [19]. That is, it is not allowed to deduce a tuple from its own negation. Stratifiability is a pure syntactic check on the rule structure and can be done prior to query processing. We remark that the combined complexity (in terms of the size of both the data and the rules) for Datalog programs with a single, recursive, non-linear rule is known to be EXPTIME-complete [20]. Although we cannot improve upon this worst-case bound, we argue that top-k pruning may also help to improve the runtime for many recursive queries in practice.

Recursion poses a challenging problem for any grounding algorithm. In our setting, the lineage formula of an answer could grow infinitely large if a cycle arises within the rules. Thus, we develop a theorem ensuring the finiteness of a lineage formula Φ, without altering the possible worlds that satisfy Φ. We formally define a *cycle* to be a subgoal $R^\gamma(\bar{X})$ whose SLD expansion results in a formula Φ containing subgoals $R^{\gamma_1}(\bar{X}_1), \dots,\ R^{\gamma_n}(\bar{X}_n)$, such that $(\gamma_1, \bar{X}_1), \dots, (\gamma_n, \bar{X}_n)$ bind or contain the same constants as $(\gamma, \bar{X})$, but the names of the unbound variables in the $\bar{X}_i$'s may differ.

Theorem 3: Let $\Phi = \Psi R^\gamma(\bar{X})\Psi'$ be a lineage formula and let the expansion $\Phi_{ex}(R^\gamma(\bar{X}))$ of $R^\gamma(\bar{X})$ yield the cycle $\Psi\Phi_{ex}(R^\gamma(\bar{X}))\Psi'$. Then it holds that:

$$\Psi\Phi_{ex}(R^\gamma(\bar{X}))\Psi' \equiv \Psi\Phi_{ex}(\Phi_{ex}(R^\gamma(\bar{X})))\Psi'$$

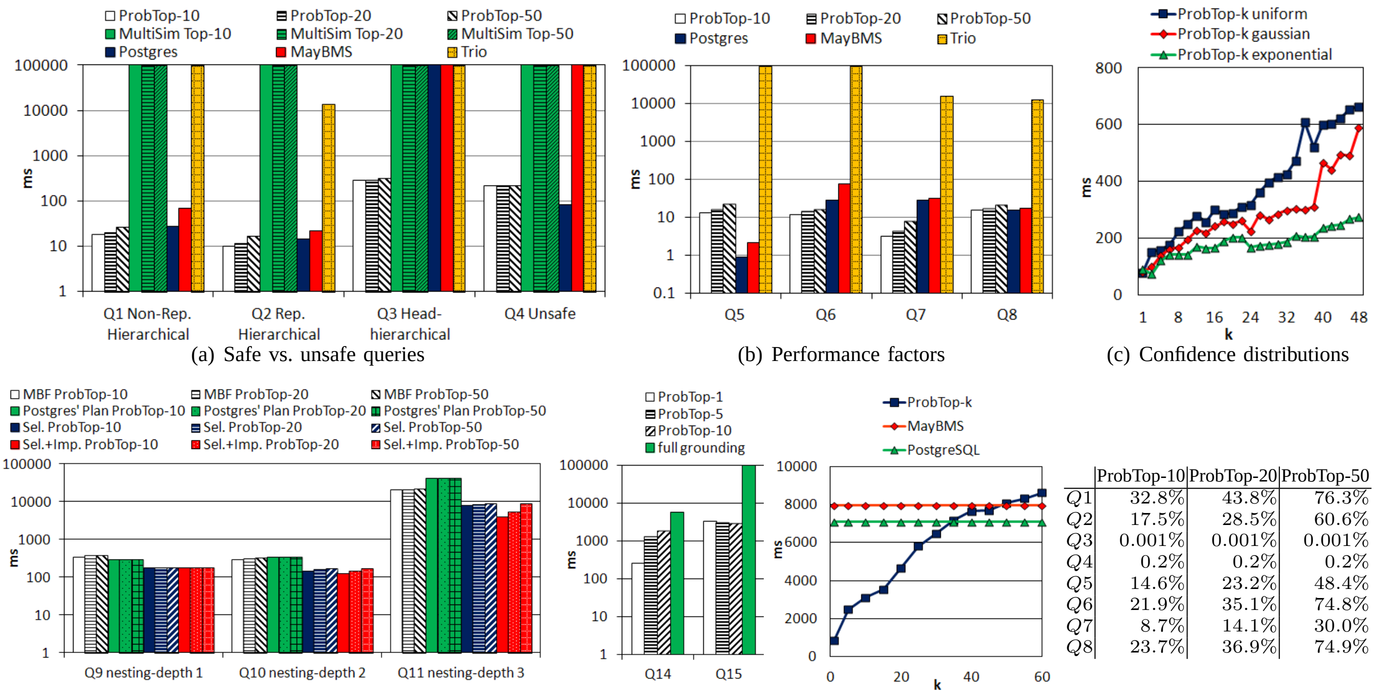
In other words, expanding a cycle more than once does not change the validity of a lineage formula, which agrees with earlier results in the context of probabilistic Datalog [27].

Proof: W.l.o.g., we assume all formulas to be in prenex form. Furthermore, let $\Phi' \vee \bigvee_i(\Phi_i'' \wedge R^\gamma(\bar{X}))$ be the DNF of $\Phi_{ex}(R^\gamma(\bar{X}))$. That is Φ' is a DNF formula, Φ_i are conjunctions of literals and both do not contain $R^\gamma(\bar{X})$. Due to stratification, $R^\gamma(\bar{X})$ must occur positively in the above formula. Now, we can rewrite $\Psi\Phi_{ex}(\Phi_{ex}(R^\gamma(\bar{X})))\Psi'$ through the following series of algebraic transformations:

$$\begin{gathered}
\Psi\Phi_{ex}(\Phi_{ex}(R^\gamma(\bar{X})))\Psi' \\
\equiv \Psi\Phi' \vee \bigvee\nolimits_i(\Phi_i'' \wedge (\Phi' \vee \bigvee\nolimits_j(\Phi_j'' \wedge R^\gamma(\bar{X}))))\Psi' \\
\equiv \Psi\Phi' \vee \bigvee\nolimits_i(\Phi_i'' \wedge \Phi) \vee \bigvee\nolimits_{i,j}(\Phi_i'' \wedge \Phi_j'' \wedge R^\gamma(\bar{X}))\Psi' \\
\equiv \Psi\Phi' \vee \bigvee\nolimits_{i,j}(\Phi_i'' \wedge \Phi_j'' \wedge R^\gamma(\bar{X}))\Psi' \\
\equiv \Psi\Phi' \vee \bigvee\nolimits_i(\Phi_i'' \wedge \Phi_i'' \wedge R^\gamma(\bar{X})) \vee \bigvee\nolimits_{i \neq j}(\Phi_i'' \wedge \Phi_j'' \wedge R^\gamma(\bar{X}))\Psi' \\
\equiv \Psi\Phi' \vee \bigvee\nolimits_i(\Phi_i'' \wedge R^\gamma(\bar{X})) \vee \bigvee\nolimits_{i \neq j}(\Phi_i'' \wedge \Phi_j'' \wedge R^\gamma(\bar{X}))\Psi' \\
\equiv \Psi\Phi' \vee \bigvee\nolimits_i(\Psi_i'' \wedge R^\gamma(\bar{X})) \\
\equiv \Psi\Phi_{ex}(R^\gamma(\bar{X}))\Psi'
\end{gathered}$$

This again yields the form of the first expansion of $R^\gamma(\bar{X})$. ■

In our implementation of SLD resolution with dynamic subgoal scheduling, we block those subgoals that form the root of a cycle in our priority queue. If all subgoals in the lineage formula of an answer are blocked, no new results can be obtained, and we replace their lineages by *false*.



(a) Safe vs. unsafe queries

(b) Performance factors

(c) Confidence distributions

(d) Subgoal scheduling

(e) Recursive rules

(f) Table materialization

	ProbTop-10	ProbTop-20	ProbTop-50
$Q1$	32.8%	43.8%	76.3%
$Q2$	17.5%	28.5%	60.6%
$Q3$	0.001%	0.001%	0.001%
$Q4$	0.2%	0.2%	0.2%
$Q5$	14.6%	23.2%	48.4%
$Q6$	21.9%	35.1%	74.8%
$Q7$	8.7%	14.1%	30.0%
$Q8$	23.7%	36.9%	74.9%

(g) Percentage of base tuples scanned

Fig. 2. Experiments

VII. Experiments

We performed our experiments on an 8-core Intel Xeon 2.4 GHz with 48 GB of RAM. Our probabilistic top-k algorithm (coined *ProbTop-k*) is implemented in Java and utilizes a PostgreSQL DB as storage backend. We employ two well-known PDB engines for comparison purposes, *MayBMS*[2] and *Trio*[3], both computing all answers and their probabilities. Additionally, we implemented the multi-simulation algorithm of [5], which we refer to as *MultiSim*. We also include comparisons against a purely deterministic DB, denoted as *PostgreSQL*, by storing confidence values in all relations but omitting the actual confidence computations. The *PostgreSQL* baseline thus serves also as a lower bound for any probabilistic top-k approach, including [5], [6], that requires full data materialization (and lineage tracing for intensional query evaluations).

A. Data Sets, Confidence Distributions, and Queries

We use two different datasets based on IMDB and YAGO. The IMDB movie dataset consists of the 6 relations *directed*, *acted*, *edited*, *produced*, *written*, and *hasCategory* summing up to $26 \cdot 10^6$ tuples. Since this data is deterministic, we sample confidence values from three synthetic distributions, namely *uniform*, *Gaussian*, and *exponential* to instantiate our probabilistic relations. Our second dataset is derived from the YAGO [28] knowledge base with $132 \cdot 10^6$ tuples, where we also sample confidences using a uniform distribution. We consider 15 different query patterns $Q1$–$Q15$, which we each instantiate into up to 1,000 individual queries by inserting randomly chosen constants into the query literals, each ensuring at least 50 answers.[4] We report average runtimes over warm disk caches by running each query 4 times in a row, and report the average runtime of the latter 3 runs. For presentation purposes, we depict runtimes of only up to 100 seconds for all systems.

B. Results

Safe vs. Unsafe Queries. We first focus on four established query classes [1], [2], [6] in PDBs, namely *non-repeating hierarchical*, *repeating hierarchical*, *non-repeating head-hierarchical*, and *general unsafe* queries, which are represented by query patterns $Q1$, $Q2$, $Q3$ and $Q4$, respectively. Only the first class is guaranteed to yield safe query plans while the latter are unsafe. Each query pattern is instantiated into 1,000 different queries. The run-times for the IMDB dataset with uniform confidences are shown in Fig. 2(a). Additionally, the table in Fig. 2(g) depicts the fraction of base tuples our top-k approach reads in comparison to the number of base tuples necessary for computing all answers. For the non-repeating hierarchical queries ($Q1$), our top-k approach outperforms all systems including the deterministic one. We mainly benefit because by far not all base tuples need to be scanned, and confidences can be computed extensionally. $Q2$ contains repeated relations and the gains in data computations are partially diminished by the Shannon expansions needed for computing the bounds. $Q3$ includes expensive data computations caused by a subquery that is shared among all answers. Since the

[2] MayBMS: http://maybms.sourceforge.net/
[3] Trio: http://infolab.stanford.edu/trio/

[4] All details on the query patterns and views can be found in [29]. $Q12$ and $Q13$ have been omitted due to space constraints.

subquery is not required to rank the results, our approach reads only very few base tuples (Fig. 2(g)). Here, our top-k algorithm successfully terminates and even outperforms the deterministic *PostgreSQL* baseline. $Q4$ contains both a subquery with expensive confidence computations as well as a subquery with major data computations. However, we can prune answers even before the expensive subqueries are fully evaluated.

Performance Factors. We next highlight the different factors that impact how our top-k approach performs against the competitors. Fig. 2(b) depicts runtimes on the IMDB dataset for four additional query patterns $Q5$, $Q6$, $Q7$, and $Q8$, which are again instantiated into $1,000$ queries each. $Q5$ yields exactly one proof for each answer candidate and no pruning in terms of omitting a proof is possible for ProbTop-k. Also, the proof involves an existentially quantified variable which limits the use of sorted input lists. As a result, our system computes most answers and *MayBMS*'s bottom-up grounding and confidence computation of all answers is much more efficient. In $Q6$, the possibility of three proofs per answer enables pruning and the lack of existential quantifiers puts our approach in favor of the others. $Q7$ is a join of two existential relations and shows where sorted input lists can provide a significant performance gain. Finally, in $Q8$ each answer has up to three proofs, however the joined relations overlap and thus require Shannon expansions. Since we repeatedly invoke these expansions to determine the bounds, the advantages of top-k pruning even out with the competitors.

Confidence Distributions. We so far focused on a uniform distribution of confidences. We now explore how different distributions can affect our performance by comparing uniform, Gaussian, and exponential. Fig. 2(c) shows the results for a join query on two existential relations over IMBD. As k grows, uniform yields the highest increase in runtime while exponential has the slowest grow, since only few tuples have high confidences. Gaussian shows a jump at $k = 40$ as more answer candidates with similar probabilities are found.

Subgoal Scheduling. In Fig. 2(d), we evaluate our scheduling techniques based on selectivity estimation (*Sel.*) and impact (*Imp.*). Our first baseline for dynamic subgoal scheduling, called "most-bound-first" (*MBF*) (aka. "bound-is-easier" [19]), chooses the subgoal with the maximum number of arguments bound at each SLD grounding step. For the second baseline, we obtained PostgreSQL's static query plan for these query patterns and forced our system to adhere to this plan (denoted as *Postgres' Plan*). Using the YAGO dataset, the three query patterns $Q9$, $Q10$, and $Q11$ were instantiated by 100 constants each. We order the query patterns by increasing nesting depth of their subqueries, such that $Q9$, $Q10$, and $Q11$ come with nesting depths of 1, 2, and 3, respectively. For $Q9$, *MBF* is outperformed by both *Postgres' Plan* and our scheduler using selectivity estimates (*Sel.*). Here, adding the impact calculations to the selectivity estimation does not yield any performance gains, but even results in slight losses. However, when moving to the higher nesting depths of $Q10$ and $Q11$, the impact calculations start improving the performance of the selectivity and impact based scheduler.

Recursive Rules. Fig. 2(e) depicts how our top-k approach performs over the YAGO data set using the recursive query patterns $Q14$ and $Q15$, which were instantiated to 50 queries each. We also include the runtime for a *full grounding* approach corresponding to an SLD grounding algorithm with lineage tracing, but without any confidence computations. $Q14$ computes ancestors of persons utilizing the *hasChild* relation, whereas $Q15$ asks for politicians of nations by transitively following the *hasSuccessor* relation. For $Q14$, the runtime increases with k, since more ancestors being generations away from the queried person have to be computed. For $Q15$ our top-k algorithm takes the same amount of time for all k's, since more than 10 politicians are known per country and are ranked in the top-10 results. For the full grounding, the lineage computation of all answers becomes very expensive.

Table Materialization. Last, we compare our top-k approach against a full materialization of all answers performed by both *MayBMS* and *PostgreSQL*. We focus on how k affects runtime. The query asks for directors of comedies (a join between the *Directed* and *hasCategory* relations with uniform confidences). Our top-k system computes the top-ranked answers for different k's. In contrast, *MayBMS* and *PostgreSQL* fully materialize a table containing all results (*PostgreSQL* ignores the confidence computation). When k is below 50, our top-k approach outperforms the others, but for larger values, the bookkeeping overhead starts dominating.

VIII. Related Work

The increasing amount of uncertain data that has become available practically at Web-scale has driven the development of various PDB engines in recent years, including systems like MystiQ [7], Trio [12], MayBMS [16], Orion [18], PrDB [30] and SPROUT [17]. Works on intensional query evaluation such as [1], [11], [12], [13], [14] capture the lineage of derived tuples as propositional formulas and have been shown to be closed and complete under the relational model. To cope with the challenge of confidence computations, recent work has concentrated on exploiting *safe query plans* [2] and *read-once formulas* [4]. In [2], Dalvi and Suciu define a dichotomy of query plans for which confidence computations can be done either in polynomial time or are #$\mathcal{P}$-hard.

As an alternative way of addressing confidence computations in PDBs, top-k style pruning approaches [5], [6], [26], [31], [32] have also been proposed. In relational DBs, the most influential work for extensional data still is given by the family of threshold algorithms by Fagin et al. [9]. A comprehensive survey of top-k queries for relational DBs is found in [10]. Most top-k approaches in the context of PDBs consider separate numerical attributes for capturing the confidence and the score of tuples, where usually only the latter is used for ranking. Soliman et al. [31] were the first to discuss the different semantics, under which one can interpret uncertain top-k queries, and thus defined *U-topK* queries and *U-kRanks* queries. Recently, Ge et al. [32] studied the tradeoffs between reporting tuples of a high score and tuples of a high probability, while Li et al. [26] proposed a unified ranking approach by considering both the scores and the confidences.

Very few works however consider top-k ranking by the marginal probabilities of query answers. Ré et al. [5] compute the top-k answers using MCMC-style sampling techniques. Recently, Olteanu and Wen [6] have further developed the idea of decomposing propositional formulas for deriving bounds based on a combination of partially expanded OBDDs and shared query plans, which can be exploited by top-k algorithms for early candidate pruning. While our bounding approach for propositional formulas is related to [6], we more generally consider bounds for first-order lineage formulas, thus having a focus on the case when views are not materialized. In [33], static probability thresholds are incorporated into the query algebra, allowing for early pruning of low confidence tuples. However, their approach does not support full relational algebra with duplicate elimination. For computing bounds on confidences of lineage formulas, there are four major works [8], [22], [23], [24], which we build upon for the propositional lineage case. However, we found the consideration of first-order lineage formulas to be a key to also incorporate pruning techniques known from managing extensional data [9] into a PDB setting. Very recently, MarkoViews [34] have been proposed, which allow for encoding complex tuple correlations via views and provide an interesting translation from Markov Logic Networks [35] to unions of conjunctive queries [1] over a tuple-independent PDB for query evaluation.

IX. Conclusions

We presented efficient processing strategies for probabilistic top-k queries which lie at the intersection of probabilistic databases and probabilistic Datalog. Our approach does not assume safe query plans nor read-once lineage formulas, and it is able to return the exact top-k answers according to their marginal probabilities in many cases when exact confidence computations for these answers are intractable. Moreover, by focusing on non-materialized views, our pruning strategies can effectively help to avoid extensive data materialization and thus can contribute to significantly reduced data computation and storage costs. Extensions of our framework allow us to adopt top-k pruning strategies and sequential access patterns known from managing extensional data, and they even help to improve the runtime for recursive rules. In future work, our methods could be combined with techniques from [6], which we expect to enable even better performance gains.

References

[1] D. Suciu, D. Olteanu, C. Ré, and C. Koch, *Probabilistic Databases*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[2] N. Dalvi and D. Suciu, "The dichotomy of conjunctive queries on probabilistic structures," in *PODS*, 2007, pp. 293–302.

[3] D. Roth, "On the hardness of approximate reasoning," *Artif. Intell.*, vol. 82, pp. 273–302, 1996.

[4] P. Sen, A. Deshpande, and L. Getoor, "Read-once functions and query evaluation in probabilistic databases," *PVLDB*, vol. 3, no. 1, pp. 1068–1079, 2010.

[5] C. Ré, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic data," in *ICDE*, 2007, pp. 886–895.

[6] D. Olteanu and H. Wen, "Ranking query answers in probabilistic databases: Complexity and efficient algorithms," in *ICDE*, 2012, pp. 282–293.

[7] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu, "MYSTIQ: a system for finding more answers by using probabilities," in *SIGMOD*, 2005, pp. 891–893.

[8] R. Fink and D. Olteanu, "On the optimal approximation of queries using tractable propositional languages," in *ICDT*, 2011, pp. 174–185.

[9] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, 2003.

[10] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, pp. 11:1–11:58, 2008.

[11] N. Fuhr, "Probabilistic Datalog - a logic for powerful retrieval methods," in *SIGIR*, 1995, pp. 282–290.

[12] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom, "Databases with uncertainty and lineage," *VLDB J.*, vol. 17, no. 2, pp. 243–264, 2008.

[13] A. D. Sarma, M. Theobald, and J. Widom, "Exploiting lineage for confidence computation in uncertain and probabilistic databases," in *ICDE*, 2008, pp. 1023–1032.

[14] P. Buneman and W. C. Tan, "Provenance in databases," in *SIGMOD*, 2007, pp. 1171–1173.

[15] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," *Theor. Comput. Sci.*, vol. 78, no. 1, pp. 159–187, 1991.

[16] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *ICDE*, 2008, pp. 983–992.

[17] D. Olteanu, J. Huang, and C. Koch, "Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases," in *ICDE*, 2009, pp. 640–651.

[18] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah, "Orion 2.0: native support for uncertain data," in *SIGMOD*, 2008, pp. 1239–1242.

[19] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[20] G. Gottlob and C. H. Papadimitriou, "On the complexity of single-rule datalog queries," *Inf. Comput.*, vol. 183, no. 1, pp. 104–122, 2003.

[21] A. Kimmig, G. V. den Broeck, and L. D. Raedt, "An algebraic Prolog for reasoning about possible worlds," in *AAAI*, 2011.

[22] D. Olteanu, J. Huang, and C. Koch, "Approximate confidence computation in probabilistic databases," in *ICDE*, 2010, pp. 145–156.

[23] Y. Sagiv and M. Yannakakis, "Equivalences among relational expressions with the union and difference operators," *J. ACM*, vol. 27, pp. 633–655, 1980.

[24] R. Fink, D. Olteanu, and S. Rath, "Providing support for full relational algebra in probabilistic databases," in *ICDE*, 2011, pp. 315–326.

[25] B. Kanagal, J. Li, and A. Deshpande, "Sensitivity analysis and explanations for robust query evaluation in probabilistic databases," in *SIGMOD*, 2011, pp. 841–852.

[26] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," *PVLDB*, vol. 2, no. 1, pp. 502–513, 2009.

[27] T. Rölleke and N. Fuhr, "Probabilistic reasoning for large scale databases," in *BTW*, 1997, pp. 118–132.

[28] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *WWW*, 2007, pp. 697–706.

[29] M. Dylla, I. Miliaraki, and M. Theobald, "Top-k query processing in probabilistic databases with non-materialized views," Research Report MPI-I-2012-5-002, 2012.

[30] P. Sen, A. Deshpande, and L. Getoor, "PrDB: managing and exploiting rich correlations in probabilistic databases," *VLDB J.*, vol. 18, no. 5, pp. 1065–1090, 2009.

[31] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007.

[32] T. Ge, S. B. Zdonik, and S. Madden, "Top-k queries on uncertain data: on score distribution and typical answers," in *SIGMOD*, 2009, pp. 375–388.

[33] Y. Qi, R. Jain, S. Singh, and S. Prabhakar, "Threshold query optimization for uncertain data," in *SIGMOD Conference*, 2010, pp. 315–326.

[34] A. Jha and D. Suciu, "Probabilistic databases with MarkoViews," to appear in PVLDB 2012.

[35] M. Richardson and P. Domingos, "Markov Logic Networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.