Dissociation and Propagation for Efficient Query Evaluation over Probabilistic Databases

Wolfgang Gatterbauer, Abhay K. Jha, Dan Suciu

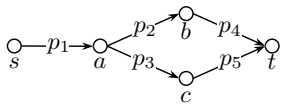
University of Washington, Seattle WA
{gatter,abhaykj,suciu}@cs.washington.edu

Abstract. Queries over probabilistic databases are either *safe*, in which case they can be evaluated entirely in a relational database engine, or *unsafe*, in which case they need to be evaluated with a general-purpose inference engine at a high cost. We propose a new approach by which *every query* is evaluated inside the database engine, by using a new method called *dissociation*. A dissociated query is obtained by adding extraneous variables to some atoms until the query becomes safe. We show that the probability of the original query and that of the dissociated query correspond to two well-known scoring functions on graphs, namely *graph reliability* (which is #P-hard), and the *propagation score* (which is related to PageRank and is in PTIME): When restricted to graphs, standard *query probability* is graph reliability, while the *dissociated probability* is the propagation score. We define a *propagation score for self-join-free conjunctive queries* and prove that it is always an upper bound for query reliability, and that both scores coincide for all safe queries. Given the widespread and successful use of graph propagation methods in practice, we argue for the dissociation method as a highly efficient way to rank probabilistic query results, especially for those queries which are highly intractable for exact probabilistic inference.

1 Introduction

Evaluating queries over probabilistic databases (PDBs) is hard in general. Despite important recent advances [15,20], today's approaches to query evaluation are not practical. Existing techniques either split the queries into safe and unsafe and compute efficiently only the former [8,19], or work very well on certain combinations of queries and data instances but can not offer performance guarantees in general [15,18], or use general purpose approximation techniques and are thus generally slow [14,23]. In this paper, we propose a new approach to evaluate queries over tuple-independent probabilistic databases by which every query can be evaluated efficiently. We achieve this by replacing the standard semantics based on *reliability*, with a related but much more efficient semantics based on *propagation*.

The semantics of a query over a PDB is based on *reliability* [12], which has roots in *network reliability* [6]. It is defined as the probability that a source node remains connected to a target node in a directed graph if edges fail independently



(a)

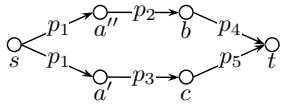
$q :- R(s,x), S(x,y), T(y,t)$

R	C	A
p_1	s	a

S	A	B
p_2	a	b
p_3	a	c

T	B	C
p_4	b	t
p_5	c	t

(b)



(c)

$q' :- R(s,x,y), S(x,y), T(y,t)$

R	C	A	B
p_1	s	a	b
p_1	s	a	c

S	A	B
p_2	a	b
p_3	a	c

T	B	C
p_4	b	t
p_5	c	t

(d)

Fig. 1. The *propagation score* $\rho(t)$ in graph (a) corresponds to the *reliability score* $r(t)$ in graph (c) with node a dissociated into a' and a''. (b,d): Corresponding chain queries q and q' with respective database instances.

with known probabilities. Computing network reliability is #P-hard. However, many successful practical applications use a semantics different from reliability, based on a *propagation scheme*. We illustrate with an example.

Example 1 (Propagation in k-partite digraphs). *Consider the 4-partite graph in Fig. 1a in which each edge i is present with independent probability p_i. The* reliability score *$r(x)$ of a node x is the probability that the source node s is connected to the node x in a randomly chosen subgraph with edges directed left-to-right. The score of interest is the reliability of a target node t:*

$$r(t) = p_1(1-(1-p_2p_4)(1-p_3p_5))$$

While reliability can be computed efficiently for series-parallel graphs as in Fig. 1a, it is #P-hard in general, even on 4-partite graphs [6]. The probability of a query over a PDB *corresponds precisely to network reliability. For example, in the case of a 4-partite graph, reliability is given by the probability of the chain query $q :- R(s,x), S(x,y), T(y,t)$ over the* PDB *shown in Fig. 1b (we use interchangeably the terms* query reliability *and* query probability *in this paper).*

In contrast, the propagation score *of a node x is a value that depends on the scores of its parent nodes and the probabilities of the incoming edges: $\rho(x) = 1-\prod_e\big(1-\rho(y_e)\cdot p_e\big)$, where e is the edge (y_e, x). By definition, $\rho(s) = 1$. In our example, the propagation score of the target node t is:*

$$\rho(t) = 1 - (1 - p_1p_2p_4)(1 - p_1p_3p_5)$$

Unlike reliability, the propagation score can *always* be computed efficiently, even on very large graphs. The reason is that reliability has an *intensional* semantics, while propagation is *extensional*[1] [10,21]. Variants of propagation have thus

[1] *Extensional approaches* compute the probability of any formula as a function of the probabilities of its subformulas according to syntactic rules, regardless of how those were derived. *Intensional approaches* reason in terms of possible worlds and keep track of dependencies.

been successfully used in a range of applications where an exact probability computation is not necessary. Examples include similarity ranking of proteins [28], integrating and ranking uncertain scientific data [9], trust propagation in social networks [13], search in associative networks [7], models of human comprehension [22], keyword search in databases [2], and computing web page reputation with the renowned PageRank[2] algorithm [5].

With this paper, we introduce a propagation score for queries over PDBs, describe the connection to the reliability score, and give a method to compute the propagation score for every conjunctive query without self-joins efficiently with a *standard relational database engine*. We propose the propagation score as an alternative semantics to the reliability score. While the propagation score differs from the reliability score, we prove several properties showing that it is a reasonable substitute: (*i*) the propagation score is always greater than or equal to the reliability score, (*ii*) the two are guaranteed to coincide for all safe queries, and (*iii*) the propagation score is very close to the reliability score in our experimental validation.

To the best of our knowledge, no definition of a propagation score on hypergraphs exists, and it is not obvious how to define such a score for queries which are not represented by graphs but by hypergraphs. Also, when restricted to k-partite graphs, the propagation score depends on the directionality of the graph. In Fig. 1a the propagation score from s to t is different from that from t to s. In fact, the latter coincides with the reliability score. It is unclear what this directionality corresponds to for arbitrary queries.

Main contributions. Our first main contribution is defining the propagation score for any self-join-free conjunctive query in terms of a *dissociation*. A dissociation is a rewriting of both the data and the query. On the data, a dissociation is obtained by making multiple, independent copies of some of the tuples in the database. Technically, this is achieved by extending the relational schema with additional attributes. On a query, a dissociation extends atoms with additional variables. We prove that a dissociation can only increase the probability of a query, and define *the propagation score of a query* as the minimum reliability of all dissociated queries that are *safe*. This is justified by the fact that, in a k-partite graph, the propagation score is precisely the probability of one dissociated safe query. Thus, in our definition, choosing a direction for the network flow in order to define the propagation score corresponds to choosing a particular dissociation that makes the query safe. Safe queries [8] can be evaluated efficiently on any probabilistic database, and we show that every query (safe or not) admits at least one safe dissociation. Therefore, the propagation score can be computed efficiently.

Our second main contribution is establishing a one-to-one correspondence between safe dissociations and traditional query plans. This result leads to an efficient algorithm for computing the propagation score of a query: iterate over all query plans, retain only those that are minimal in the dissociation order, evaluate

[2] Note that the propagation score is *not* the same as PageRank. However, both share the common principle that the score of a node is defined only *in terms of the scores of its neighbors*, and not in terms of the entire topology of the graph.

those on the original data, and return the minimum probability. Importantly, there is *no need to dissociate the actual data*, which is an expensive step. We give a system R-style algorithm that enumerates all plans that correspond to minimal safe dissociated queries. A so far unknown corollary of our work is that every query plan gives an upper bound on query reliability.

Example 2 (Dissociation). *We have seen that the propagation score differs from the reliability score on the DAG in Fig. 1a. By inspecting the expressions of the two scores, one can see that they differ in how they treat p_1: reliability treats it as a single event, while propagation treats it as two independent events. In fact, the propagation score is precisely the reliability score of the DAG in Fig. 1c, which has two copies of p_1. We call this DAG the* dissociation *of the DAG in Fig. 1a. At the level of the database instance, dissociation can be obtained by adding a new attribute B to the first relation R (Fig. 1d). The dissociated query is $q^B :- R(s, x, y), S(x, y), T(y, t)$, and one can check that its probability is indeed the same as the propagation score for the data in Fig. 1a. The important observation here is that, while the evaluation problem for q is #P-hard because it is an unsafe query [8], the query q^B is safe and can therefore be computed efficiently.*

A query q usually has more than one dissociation: q has a second dissociation $q^A :- R(s, x), S(x, y), T(x, y, t)$ obtained by adding the attribute A to T (not shown in the figure). Its probability corresponds to the propagation score from t to s, i.e. from right to left. And there is a third dissociation, $q^{BA} :- R(s, x, y)$, $S(x, y), T(x, y, t)$. We prove that each dissociation step can only increase the probability, hence $r(q) \leq r(q^B) \leq r(q^{BA})$ and $r(q) \leq r(q^A) \leq r(q^{BA})$. We define the propagation score of q as the smallest probability of all dissociations. The database system has to compute $r(q^B)$ and $r(q^A)$ and return the smallest score: on the graph in Fig. 1a this is $r(q^A)$, since $r(q) = r(q^A)$.

Outline. We review basic definitions (Sect. 2), then formally introduce query dissociation and the propagation score (Sect. 3). We prove its strong connection to query plans (Sect. 4), describe our experimental evaluation (Sect. 5), review related work (Sect. 6), before we conclude (Sect. 7). Details, all proofs, more experiments and several optimizations are covered in the technical report [11].

2 Preliminaries

We consider PDBs where each tuple t has a probability $p(t) \in [0, 1]$. We denote with D the database instance, i.e. the collection of tuples and their probabilities. A *possible world* is generated by independently including each tuple t in the world with probability $p(t)$. Thus, the database D is *tuple-independent*. Consider a Boolean conjunctive query q

$$q :- g_1, \ldots, g_m$$

where $g_1, \ldots, g_m$ are relational atoms, sometimes called subgoals, over a vocabulary $R_1, \ldots, R_k$. The focus of probabilistic query evaluation is to compute $\mathbf{P}[q]$,

which is the probability that the query is true in a randomly chosen world, and which we refer to as the *query reliability* $r(q)$ [12].

It is known that the data complexity of a conjunctive query q is either PTIME or #P-hard [8]. The class of PTIME queries, sometimes called *safe queries*, is best understood in the case of Boolean queries without self-joins, keys and deterministic relations. We will focus on this important case in this paper. With this restriction, the safe queries are precisely the *hierarchical queries*:

Definition 1 (Hierarchical queries [8]). *For every variable x in q, denote $sg(x)$ the set of subgoals that contain x. Then q is called hierarchical if for any two variables x, y, one of the following three conditions hold: $sg(x) \subseteq sg(y)$, $sg(x) \cap sg(y) = \emptyset$, or $sg(x) \supseteq sg(y)$.*

For example, the query $q :- R(x,y), S(y,z), T(y,z,u)$ is hierarchical, while $q :- R(x,y), S(y,z), T(z,u)$ is not. It is known that every hierarchical query can be computed in PTIME, and every non-hierarchical query is #P-hard.

We next introduce a query plan for a conjunctive query:

Definition 2 (Query plan P). *Let $R_1, \ldots, R_m$ be a relational vocabulary. A query plan, or simply plan, is given by the grammar*

$$P ::= R_i(\bar{x}) \mid \pi_{\bar{x}} P \mid \bowtie\big[P_1, \ldots, P_k\big]$$

where $R_i(\bar{x})$ is a relational atom containing the variables $\bar{x}$ and constants, $\pi_{\bar{x}}$ is the project operator with duplicate elimination, and $\bowtie\big[\ldots\big]$ is the natural join, which we allow to be k-ary, for $k \geq 2$. We require that joins and projections alternate in a plan. We do not distinguish between join orders, i.e. $\bowtie\big[P_1, P_2\big]$ is the same as $\bowtie\big[P_2, P_1\big]$.

Denote q_P the query consisting of all atoms mentioned in (sub-)plan P. We define the *head variables* $\texttt{HVar}(P)$ inductively as

$$\texttt{HVar}(R_i(\bar{x})) = \bar{x}$$
$$\texttt{HVar}(\pi_{\bar{x}}(P)) = \bar{x}$$
$$\texttt{HVar}(\bowtie\big[P_1, \ldots, P_k\big]) = \bigcup_{i=1}^{k} \texttt{HVar}(P_i)$$

A plan is called Boolean if $\texttt{HVar}(P) = \emptyset$. We assume the usual sanity conditions on plans to be satisfied: in a project operator $\pi_{\bar{x}}(P)$ we assume $\bar{x} \subseteq \texttt{HVar}(P)$, and each variable y is projected away in at most one project operator, i.e. there exists at most one operator $\pi_{\bar{x}}(P)$ s.t. $y \in \texttt{HVar}(P) - \bar{x}$.

A plan P is evaluated on a probabilistic database D using an *extensional semantics* [10], [21, p. 3]: Each subplan P returns an intermediate relation of arity $|\texttt{HVar}(P)| + 1$. The extra attribute stores the probability of each tuple. To compute the probability, each operator assumes the input tuples to be *independent*, i.e. the probabilistic join operator $\bowtie^p\big[\ldots\big]$ multiplies the tuple probabilities $\Pi_i p_i$, and the probabilistic project operator with duplicate elimination π^p computes the probability as $1 - \Pi_i(1 - p_i)$. For a Boolean plan P, this results in a

single probability value, which we denote $score(P)$. In general, this is not the correct query reliability $r(q_P)$, which, recall, is defined in terms of possible worlds: $score(P) \neq r(q_p)$.

Definition 3 (Safe plan). *A plan P is called* safe *if for any join operator $\bowtie^p[P_1, \ldots, P_k]$ the following holds: $\texttt{HVar}(P_i) = \texttt{HVar}(P_j)$, for all $1 \leq i, j \leq k$.*

The following are well known facts about safe queries and safe plans.

Proposition 1 (Safety). *(1) Let P be a plan for the conjunctive query without self-joins q_P. Then P is safe iff for any probabilistic database, $score(P) = r(q_p)$. (2) Let q be a conjunctive query. Then the following are equivalent: q is safe; q is hierarchical; q admits a safe plan. Moreover, the safe plan is unique (up to reordering of the operands in join operators).*

Example 3 (Safe plan). *An example safe query and its unique safe plan:*

$$q :- R(x,y), S(y,z), T(y,z,u)$$
$$P = \pi^p_\emptyset \bowtie^p [\pi^p_y R(x,y), \pi^p_y \bowtie^p [S(y,z), \pi^p_{y,z} T(y,z,u)]]$$

3 Dissociation and Propagation

In this section, we define the technique of *query dissociation* and the semantics of *propagation score*. We first define the approach formally, then describe in Sect. 4 an efficient method to evaluate propagation and illustrate with examples. All proofs and more details are provided in the appendix.

In the following, we write $\texttt{Var}(q)$ for set of variables in the body of a query q, $\texttt{Var}(g_i)$ for the variables in a subgoal g_i, and A for the active domain of a database D. We use the bar sign (e.g. $\bar{x}$) to denote both sets or tuples.

Definition 4 (Query dissociation). *A dissociation of a conjunctive query $q :- g_1, \ldots, g_m$ is a collection of sets of variables $\Delta = \{\bar{y}_1, \ldots, \bar{y}_m\}$ with $\bar{y}_i \subseteq \texttt{Var}(q) - \texttt{Var}(g_i)$.*

Definition 5 (Table dissociation). *Given a query $q :- g_1, \ldots, g_m$ with $g_i = R_i(\bar{x}_i)$, the active domain A, and a query dissociation $\Delta = \{\bar{y}_1, \ldots, \bar{y}_m\}$. The dissociation of table R_i on $\bar{y}_i = (y_{i1}, \ldots, y_{ik})$ is the relation given by query*

$$R_i^{\bar{y}_i}(\bar{x}_i, \bar{y}_i) :- R_i(\bar{x}_i), A(y_{i1}), \ldots, A(y_{ik})$$

Conceptually, a *dissociation of a table* is the multi-cross product with the active domain so that each tuple in the original table is copied to multiple tuples in the dissociated table. Recall that each tuple in the original table represents an independent probabilistic event. The dissociated table now contains multiple copies of each tuple, all with the same probability, yet considered to represent *independent* events. Thus, the dissociated table has a different probabilistic semantics than the original table.

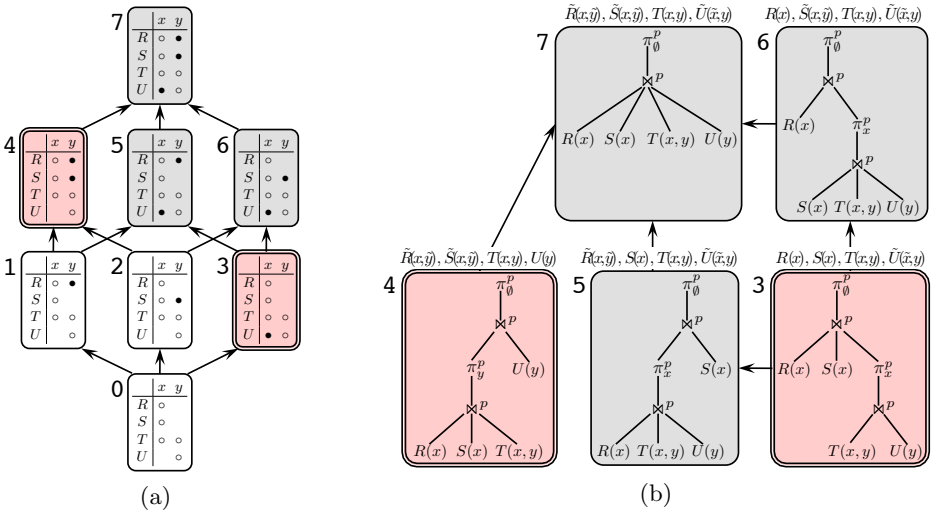


Fig. 2. (a): Partial dissociation order of $q :- R(x), S(x), T(x,y), U(y)$. Safe dissociations are shaded $(3,4,5,6,7)$, *minimal safe dissociations* are shaded in red and double-lined $(3,4)$. (b): All 5 possible query plans, their partial order and correspondence to safe dissociations in the partial dissociation order of q.

Definition 6 (Query dissociation semantics). *Let $q :- R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$ be a query and $\Delta = \{\bar{y}_1, \ldots, \bar{y}_m\}$ a dissociation for q. The dissociated query is:*

$$q^\Delta :- R_1^{\bar{y}_1}(\bar{x}_1, \bar{y}_1), \ldots, R_m^{\bar{y}_m}(\bar{x}_m, \bar{y}_m)$$

Thus, query dissociation works as follows: Add some variables to some atoms in the query. This results in the *dissociated query* over a new schema[3]. Transform the probabilistic database by replicating some of their tuples and by adding new attributes to match the new schema. This is the *dissociated database*. Finally, compute the probability of the dissociated query on the dissociated database[4].

Our first major technical result in this paper shows that query dissociation can only increase the probability. We state it in a slightly more general form, by noting that the set of dissociations forms a partial order.

Definition 7 (Partial dissociation order). *We define the partial order on the dissociations of a query as:*

$$\Delta \succeq \Delta' \;\Leftrightarrow\; \forall i : \bar{y}_i \supseteq \bar{y}'_i$$

Theorem 2 (Partial dissociation order). *For every two dissociations Δ and Δ' of a query q, the following holds over every database instance:*

$$\Delta \succeq \Delta' \;\Leftrightarrow\; r(q^\Delta) \geq r(q^{\Delta'})$$

[3] Note that several alternative dissociations are possible, in general.

[4] Note that this is the *semantics* of a dissociated query, and not the way we actually evaluate queries. In Sect. 4 we give a method that evaluates the dissociated query without actually modifying the tables in the database.

Corollary 3 (Upper query bounds). *For every database and every dissociation Δ of a query q: $r(q^\Delta) \geq r(q)$.*

Corollary 3 immediately follows from Theorem 2 as every query is a dissociation of itself. The total number of dissociations corresponds to the cardinality of the power set of variables that can be added to tables. Hence, for every query with n non-head variables and m clauses, there are 2^K possible dissociations with $K = mn - k$ and $k = \sum_{i=1}^{m} |\mathtt{Var}(g_i)|$ forming a partial order in the shape of a power set (see Fig. 2a).

We next define *the propagation score of a query* as the minimum probability of all those dissociations in the partial dissociation order that are *safe*.

Definition 8 (Safe dissociation). *A dissociation Δ_s of a query q is called safe if the dissociated query q^{Δ_s} is safe.*

Definition 9 (Propagation score). *The propagation score $\rho(q)$ for a query q is the minimum of the scores of all safe dissociations: $\rho(q) = \min_{\Delta_s} r(q^{\Delta_s})$.*

We propose to adopt the propagation score as an alternative semantics to reliability. While computing the reliability $r(q)$ is #P-hard in general, computing the propagation score $\rho(q)$ is always in PTIME in the size of the database. Further, $\rho(q) \geq r(q)$ and, if q is safe, then $\rho(q) = r(q)$ (both claims follow immediately from Corollary 3).

We now justify our definitions of dissociation and query propagation: When a graph is k-partite, then its reliability can be expressed by a conjunctive chain query q. Further, the propagation score over this graph corresponds to exactly one of several possible dissociations of this query q. *Query dissociation is thus a strict generalization of graph propagation on k-partite graphs.* And we define query propagation as corresponding to the dissociation with minimum reliability.

Proposition 4 (Connection propagation score). *Let $G = (V, E)$ be a k+1-partite digraph with a source node s and a target node t, where each edge has a probability. The nodes are partitioned into $V = \{s\} \cup V_2 \cup \ldots \cup V_k \cup \{t\}$, and the edges are $E = \bigcup_i R_i$, where R_i denotes the set of edges from V_i to V_{i+1} with $i \in \{1, \ldots, k\}$. Then:*

(a) *The network reliability of the graph is $r(q)$, where:*

$$q :- R_1(s, x_1), R_2(x_1, x_2), \ldots, R_k(x_{k-1}, t)$$

(b) *Using $\bar{x}_{[i,j]}$ as short form for $x_i, \ldots, x_j$, the propagation score (as defined in Example 1) is $r(q^\Delta)$, where:*

$$q^\Delta :- R_1(s, \bar{x}_{[1,k-1]}), R_2(x_{[2,k-1]}), \ldots, R_k(x_{k-1}, t)$$

4 Dissociations and Plans

Thus, in order to compute the propagation score of a query q, we need to compute several dissociated safe queries q^Δ. For that, we will not apply naively Def. 6,

because the table dissociation part (Def. 5) computes several cartesian products, and is very inefficient. Instead, we describe here an approach for computing $r(q^{\Delta})$ without dissociating either the query or the tables. The second major technical result of our paper allows us to perform dissociation very efficiently.

Theorem 5 (Safe dissociation). *Let q be a conjunctive query without self-joins. There exists an isomorphism between safe dissociations Δ_s of q and query plans P for q. Moreover, the reliability of the dissociated query is equal to the score of the plan, $r(q^{\Delta_s}) = score(P)$.*

We describe this isomorphism briefly. Consider a safe dissociation q^{Δ}, and denote its unique safe plan P^{Δ}. This plan uses dissociated relations, hence each relation $R_i^{\bar{y}_i}$ has some extraneous variables $\bar{y}_i$. Drop all the variables $\bar{y}_i$ from the relations, and from all operators that use them: this transforms P^{Δ} into a regular (unsafe) plan P for q. Conversely, consider a plan P for q (P is not necessarily safe; in fact if q is unsafe then there is no safe plan at all). We define its corresponding safe dissociation Δ as follows. For each join operation $\bowtie^p\big[P_1, \ldots, P_k\big]$, let the *join variables* be $\texttt{JVar} = \bigcup_j \texttt{HVar}[P_j]$: for every relation R_i occurring in P_j, add the variables $\texttt{JVar}-\texttt{HVar}[P_j]$ to $\bar{y}_i$. For example, consider the lower join in Fig. 2b box (5): $\bowtie^p\big[R(x), T(x,y), U(y)\big]$. Here $\texttt{JVar} = \{x, y\}$ and the dissociation of this subplan is $\tilde{R}(x,\tilde{y}), T(x,y), \tilde{U}(\tilde{x}, y)$, where the tilde ($\tilde{}$) indicates dissociated relations and variables. The complete safe query is shown at the top of the box (5). Note that while there is a one-to-one mapping between safe dissociations and query plans, unsafe dissociations do not correspond to plans (see Fig. 2a).

We have seen (right after Def. 2) that the extensional semantics of an unsafe plan P differs from the true reliability, $score(P) \neq r(q)$. Since we have shown that $score(P) = r(q^{\Delta})$ for some dissociation Δ, we derive the following important corollary:

Corollary 6 (Query plans as bounds). *Let P be any plan for a query q. Then $score(P) \geq r(q)$. In other words, any query plan for evaluating a query inside a database gives an upper bound to the actual query reliability $r(q)$.*

To summarize, we have now a much more efficient way to compute $\min_{\Delta}(r(q^{\Delta}))$: iterate over all plans P, and compute $\min_P score(P)$. Thus, there no need to dissociate the input tables which is very inefficient: each plan P is evaluated directly on the original probabilistic database. However, this approach is inefficient in that it computes some redundant plans: for example, in Fig. 2 plans 5, 6, 7 are redundant, since, in the dissociation order, they are all greater than plan 3. It suffices to evaluate only the *minimal query plans*, i.e. those for which the corresponding dissociation is minimal among all safe dissociations: in our example, these are plans 3 and 4.

Example 4 (Safe dissociations). *Take the query $q :- R(x), S(x), T(x,y), U(y)$. It is unsafe and $K = 4 \cdot 2 - 5 = 3$. Figure 2a shows its partial dissociation order*

2^3, *and Fig. 2b shows all 5 possible query plans. Two of those plans are minimal in the partial order of plans:*

$$q^{(3)} :- R(x), S(x), T(x,y), \tilde{U}(\tilde{x}, y)$$
$$q^{(4)} :- \tilde{R}(x,\tilde{y}), \tilde{S}(x,\tilde{y}), T(x,y), U(y)$$

The corresponding query plans over the original tables (and safe derivations over dissociated tables) are:

$$P^{(3)} = \pi^p_{\emptyset} \bowtie^p \big[R(x), S(x), \pi^p_x \bowtie^p \big[T(x,y), U(y)\big]\big]$$
$$P^{(4)} = \pi^p_{\emptyset} \bowtie^p \big[U(y), \pi^p_y \bowtie^p \big[R(x), S(x), T(x,y)\big]\big]$$

The propagation score is the minimum of the scores of all minimal plans: $\rho(q) = \min_{i\in\{3,4\}} \big[score\big(P^{(i)}\big)\big]$.

Note that "safetyzation by dissociation" is not monotone, and dissociation can also make a safe query unsafe. For example, the query $R(x), S(x,y), T(x,y,z)$ is safe, but its dissociation $\tilde{R}(x,\tilde{z}), S(x,y), T(x,y,z)$ is not.

We now describe an algorithm for enumerating all plans that correspond to minimal dissociations. Let $q :- g_1, \ldots, g_m$. For each nonempty subset $\bar{s} \subseteq \{1, \ldots, m\}$, denote $\texttt{HVar}[\bar{s}] \subseteq \texttt{Var}(q)$ to be the following set of variables:

$$\texttt{HVar}[\bar{s}] = \bigcup_{i\in\bar{s}} \texttt{Var}(g_i) \cap \bigcup_{j\notin\bar{s}} \texttt{Var}(g_j)$$

For any subplan P of some plan for q, denote $\bar{s}_P = \{i \mid g_i \in P\}$. One can easily check that, for any subplan P, $\texttt{HVar}[\bar{s}_P] \subseteq \texttt{HVar}(P)$. In other words, $\texttt{HVar}[\bar{s}]$ represents the minimal set of head variables of any subplan over the atoms in $\bar{s}$. In a minimal plan, we must have equality at each node that corresponds to a projection.

Proposition 7 (Minimal plans). *A plan corresponds to a minimal dissociation of a query iff:*

(a) *for every projection subplan* P: $\texttt{HVar}(P) = \texttt{HVar}[\bar{s}_P]$

(b) *for every join subplan* $P = \bowtie^p \big[P_1, \ldots, P_k\big]$*: if* x *is a variable s.t.* $\exists i.x \in \texttt{HVar}(P_i) - \texttt{HVar}[\bar{s}_P]$ *then* $\forall j \in \{1, \ldots, k\}$, $x \in \texttt{HVar}(P_j)$ *. In other words:* $\bigcup_i \texttt{HVar}(P_i) - \bigcap_i \texttt{HVar}(P_i) = \texttt{HVar}[\bar{s}_p]$.

The second condition says this. Suppose we have a variable x that appears in some, but not all of the operands P_i of a join. Then we could have projected it earlier, resulting in a smaller dissociation. The only case when we cannot do it is when x is needed later, i.e. $x \in \texttt{HVar}[\bar{s}_P]$.

Example 5 (Minimal plans). *We illustrate for the non-minimal plan* $P^{(5)}$ *from Fig. 2 and its subplan* $P = \bowtie^p \big[R(x), T(x,y), U(y)\big]$*. P includes subgoals R, T, and U. Here,* $\texttt{HVar}(P) = \{x, y\}$*, whereas* $\texttt{HVar}[\{R, T, U\}] = \{x\}$*. The difference is* $\{y\}$*, which should appear in any subplan of P according to Proposition 7(b). However, it does not appear in* $R(x)$*. Moving* $R(x)$ *to the later join with* $S(x)$ *makes the condition fulfilled and results in the minimal plan* $P^{(3)}$*.*

Algorithm 1: (Minimal Plans) generates all plans that correspond to a minimal safe dissociation

Input: Query $q :- g_1, \ldots, g_m$
Output: All minimal plans $\mathcal{P}(\bar{s})$ for $\bar{s} = \{1, \ldots, m\}$

foreach $i \in \{1, \ldots, m\}$ **do**
  if $\texttt{HVar}[\{i\}] = \texttt{Var}(g_i)$ **then** $\mathcal{P}[\{i\}] = \{g_i\}$
  else $\mathcal{P}[\{i\}] = \{\pi^p_{\texttt{HVar}[\{i\}]}(g_i)\}$
foreach $\bar{s} \subseteq \{1, \ldots, m\}$ *for increasing size of* $|\bar{s}|$ **do**
  Set $\mathcal{P}[\bar{s}] = \emptyset$
  foreach $k \geq 2$ *and every partition of* $\bar{s} = \bar{s}_1 \cup \ldots \cup \bar{s}_k$ **do**
    if $\bigcup_i \texttt{HVar}(P_i) - \bigcap_i \texttt{HVar}(P_i) = \texttt{HVar}[\bar{s}]$ **then**
      forall $P_1 \in \mathcal{P}[\bar{s}_1]$, $\ldots$, $P_k \in \mathcal{P}[\bar{s}_k]$ **do**
        $\mathcal{P}[\bar{s}] = \mathcal{P}[\bar{s}] \cup \{\pi^p_{\texttt{HVar}[\bar{s}]} \bowtie^p [P_1, \ldots, P_k]\}$

This proposition gives us immediately Algorithm 1 for enumerating all minimal plans, bottom up. It computes, for each non-empty subset $\bar{s} \subseteq \{1, \ldots, m\}$, the set $\mathcal{P}[\bar{s}]$ of all minimal plans over the atoms g_i, with $i \in \bar{s}$, and can use any Systems R style optimizer.

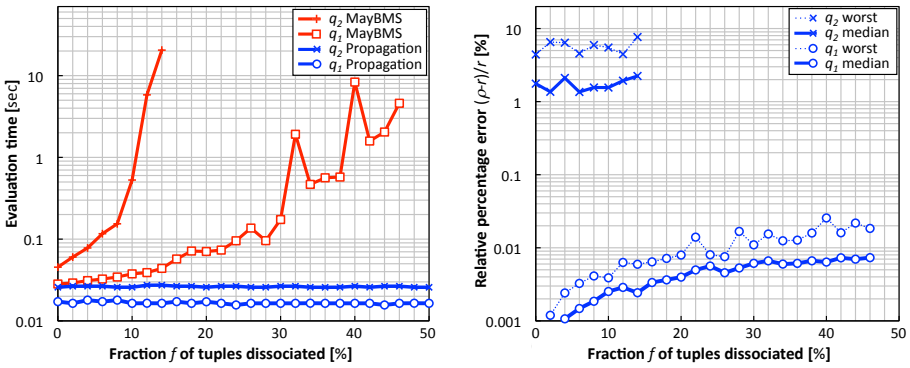
We end this section by commenting on the number of minimal safe dissociations. Not surprisingly, this number is exponential in the size of the query. To see this, consider the following query: $q :- R_1(x_1), \ldots, R_n(x_n), U(x_1, \ldots, x_n)$. There are exactly $n!$ minimal safe dissociations: Take any consistent preorder $\preceq$ on the variables. It must be a total preorder, i.e. for any i, j, either $x_i \preceq x_j$ or $x_j \preceq x_i$, because x_i, x_j occur together in U. Since it is minimal, $\preceq$ must be an order, i.e. we can't have both $x_i \preceq x_j$ and $x_j \preceq x_i$ for $i \neq j$. Therefore, $\preceq$ is a total order, and there are $n!$ such. Note that while the number of safe dissociations is exponential in the size of the query (we do not address query complexity in this paper), the number of query plans is independent of the size of the database, and hence our approach has PTIME data complexity for *all* queries [26].

In summary, our evaluation approach allows to rank answers to *every* query (safe or unsafe) in polynomial time in the size of the database, and is conservative w.r.t. the ranking according to the possible world semantics for both safe queries.

5 Experiments on Synthetic Data

We next illustrate both the the (i) efficiency and (ii) quality of the dissociation method for *increasingly intractable* queries and datasets. For timing, we compare against MayBMS [1], an existing state-of-the-art PDB system, to calculate exact probabilities. For quality, we take the results of MayBMS as ground truth for query reliability and report the relative difference between the propagation and reliability scores.

Experimental setup. ▶ *Queries*: We consider two canonical unsafe 'chain queries' $q_1 :- R(x), S(x, y), T(y)$ and $q_2 :- R(x), S_1(x, y), S_2(y, z), T(z)$. As will



(a) (b)

Fig. 3. Results for timing (a) and quality (b) experiments. See text for details.

become clear from the experiments, it is increasingly hard to get the true query reliability scores for more complex queries and datasets, and already with these two simple queries, we quickly enter the area of intractable queries and datasets. ▶ *Datasets*: Given a fraction $f \in [0, 1]$, we generate the dataset as follows: Both relations R and T contain 500 tuples each. For every tuple x in R, the functional dependency $x \to y$ on S holds with probability $1 - f$. For those x that satisfy the FD, we randomly choose a y from T and create one tuple (x, y) in S; for those x that violate the FD, we randomly choose 2 or 3 y from T to create 2 or 3 tuples in S. Hence, the parameter f is the fraction of tuples that are dissociated, and also serves as *measure of the intractability* of the query for MayBMS. Relations S_1 and S_2 are generated analogously according to FDs $x \to y$ and $y \to z$. We limit the fanout for each x to be maximum 2 or 3, as otherwise, the *treewidth* increases rapidly and, even for small fraction f, the query quickly becomes intractable for MayBMS, thus limiting the range of f we can get the ground truth for. The probability of each tuple is sampled uniformly from an interval $[0, u]$, with u chosen to create a query reliability of around 0.5. ▶ *Equipment*: The experiments are run on a Windows Server 2008 machine with Quad Core 2 GHz and 8 GB RAM. Our implementation is done in Java, wherein the program sends an SQL query to a SQL Server 2008 database. The competitor, MayBMS, is implemented in C as an extension of the PostgreSQL 8.3 backend. ▶ *Evaluation*: For each parameter f in steps of 2%, we generate 20 datasets and evaluate both MayBMS and propagation on each. For timing, we take the sum over all 20 datasets. For quality, we report both the *median* and *worst percentage error* over the 20 datasets, where percentage error is $\frac{\rho - r}{r} \cdot 100\%$, with ρ and r being propagation and reliability scores, respectively. To compute the propagation score, we execute the query using a left-right plan. It is easy to see that, using this plan, the tuples dissociated are exactly those that violate the FDs discussed above. Hence f indeed controls the fraction of tuples that are dissociated to those that could have been dissociated.

Timing experiments. Figure 3a demonstrates that the two simple queries are indeed not trivial, and evaluation with an exact solver like MayBMS quickly becomes intractable. In contrast, propagation evaluates independent of f. The results also reaffirm why we need an approach like this: exact probabilistic inference is expensive, and to scale up we need to go from intensional approaches to *effective extensional approaches* like the one proposed in this paper.

Quality experiments. Figure 3b shows the percentage error for varying fractions of dissociations. We only report data points for which MayBMS finished within 1 min, and we could get the ground truth. The error is very low for q_1 and one could argue this is because each dissociation was small; since each x is connected to at most 3 y in S. But even then, this graph demonstrates that the error doesn't go up steeply for increasing fraction f. In fact, the slope seems to decrease and it looks like the error converges. For q_2, we could not get enough data points to see the behavior as MayBMS did not finish in time. But one can see, there is an order of difference between the error rate. This isn't surprising, because here dissociation happens twice and the error is compounded. In summary, while we don't have a concrete bound of how much propagation deviates from reliability in the general case, our experiments suggest that if each tuple is dissociated only a few times, then the error rate does not increase steeply as the number of dissociations increases. The error rate may even converge. But this analysis of the exact relationship between query reliability and query propagation requires future theoretical and experimental work.

6 Related Work

Current approaches of probabilistic query evaluation either use exact methods or sampling approaches. *Exact approaches* [1,15,18,25] work well on queries with simple lineage expressions, but perform very poorly on database instances with high tree-width or long lineage expressions. *Sampling approaches* [14,16,23], on the other hand, may not be efficient even on simple queries. Some deterministic approximation algorithms [20,24] have been proposed that can approximate the query probability within any error bound, but they have no guarantee over running time. These approaches decompose a formulae recursively into independent/disjoint components via some heuristic until the approximation guarantee is within the error bound. In contrast, our approach is the only *fully extensional approach* to approximating query probabilities in probabilistic databases and therefore scales independent of the database size.

Our approach of focusing on the extensional evaluation of probabilistic databases seems vaguely related to work on *possibilistic databases* [3,4]. This body of work suggests, as foundation for quantifying uncertainty, a completely different theory, namely that of possibility theory instead of probability theory. However, we are not aware of any experimental evaluation of a possibilistic database, nor of a characterization of the exact data complexity of this model. Hence, it is difficult to compare in terms of theoretical or practical performance.

While our work is motivated by ranking query answers, it is conceptually very different from a recent number of proposals of alternative ranking semantics (see e.g. [17]). While those works are all based on the standard reliability semantics and vary in alternative ranking methods, our goal with this paper is to propose an efficient way to evaluate those scores that can be used for ranking.

Query dissociation is also related to recent work in graphical models that tries to give upper bounds on the partition function of a Markov random field. Wainwright et al. [27] develop a method to *obtain optimal upper bounds* by replacing the original distribution using a *collection of tractable distributions*, i.e. such for which the partition function can be calculated efficiently by a recursive algorithm. In our work, efficient approximations of distributions at the schema level are those that allow a *safe query plan*, and thus can be evaluated in a relational database engine. One up to now unknown corollary of our theory is that *every* query plan is an upper bound on query reliability. We further give an algorithm to find *the minimum of all instance-independent upper bounds*.

7 Conclusion

In this paper, we developed a new scoring function called *propagation* for ranking query results over probabilistic databases. Our semantics is based on a sound and principled theory of *query dissociation*, and can be evaluated efficiently in an off-the-shelf relational database engine for *any type of self-join-free conjunctive query*. We proved that query propagation is an upper bound to query reliability, that both scores coincide for safe queries, and that propagation naturally extends the case of safe queries to unsafe queries. Finally, our experiments validated that propagation is a viable alternative for evaluating intractable queries, i.e. cases of queries and database instances that cannot be handled by the current state-of-the-art in probabilistic query evaluation.

Acknowledgement. This work was supported in part by NSF grants IIS-0513877, IIS-0713576, and IIS-0915054. We thank the anonymous reviewers for helpful comments that improved the quality of presentation of this paper.

References

1. L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.
2. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
3. P. Bosc and O. Pivert. About projection-selection-join queries addressed to possibilistic relational databases. *IEEE T. Fuzzy Systems*, 13(1):124–139, 2005.
4. P. Bosc, O. Pivert, and H. Prade. A model based on possibilistic certainty levels for incomplete databases. In *SUM*, 2009.
5. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

6. C. J. Colbourn. *The combinatorics of network reliability*. Oxford University Press, New York, 1987.
7. F. Crestani. Application of spreading activation techniques in information retrieval. *Artif. Intell. Rev.*, 11(6):453–482, 1997.
8. N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.
9. L. Detwiler, W. Gatterbauer, B. Louie, D. Suciu, and P. Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In *ICDE*, 2009. (see `http://db.cs.washington.edu/propagation`).
10. N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.
11. W. Gatterbauer, A. K. Jha, and D. Suciu. Dissociation and propagation for efficient query evaluation over probabilistic databases. Technical report, UW-CSE-10-04-01, 2010. (see `http://db.cs.washington.edu/propagation`).
12. E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, 1998.
13. R. V. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *WWW*, 2004.
14. R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.
15. A. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, 2010.
16. S. Joshi and C. M. Jermaine. Sampling-based estimators for subset-based queries. *VLDB J.*, 18(1):181–202, 2009.
17. J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.
18. D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, 2008.
19. D. Olteanu, J. Huang, and C. Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, 2009.
20. D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.
21. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, San Mateo, Calif., 1988.
22. M. R. Quillian. Semantic memory. In *Semantic Information Processing*, pages 227–270. MIT Press, 1968.
23. C. Ré, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.
24. C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.
25. P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.
26. M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.
27. M. J. Wainwright, T. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.
28. J. Weston, A. Elisseeff, D. Zhou, C. S. Leslie, and W. S. Noble. Protein ranking: from local to global structure in the protein similarity network. *Proc Natl Acad Sci USA*, 101(17):6559–63, Apr 2004.