smallskip
myCap
textbf :
1.2mm1.2mm
ForAllforalldoendfch

Approximate Lifted Inference in Probabilistic Databases

Wolfgang Gatterbauer
Carnegie Mellon University
gatt@cmu.edu

Dan Suciu
University of Washington
suciu@cs.washington.edu

[From Wolfgang: add loopy belief propagation reference [?]]

ABSTRACT

This paper proposes a new approach for approximate evaluation of #P-hard queries over probabilistic databases. In our approach, every query is evaluated entirely in the database engine by evaluating a fixed number of query plans, each providing an upper bound on the true probability, then taking their minimum. We provide an algorithm that takes into account important schema information to enumerate only the minimal necessary plans among all possible plans. Importantly, this algorithm is a strict generalization of all known results of PTIME self-join free conjunctive queries: A query is safe if and only if our algorithm returns one single plan. We also apply three relational query optimization techniques to evaluate all minimal safe plans very fast. We give a detailed experimental evaluation of our approach and, in the process, provide a new way of thinking about the value of probabilistic methods over non-probabilistic methods for ranking query answers.

1. INTRODUCTION

Probabilistic inference over large data sets is becoming a central data management problem. Recent large knowledge bases, such as Yago [?], Nell [?], DeepDive [?], or Google's Knowledge Vault [?], have millions to billions of uncertain tuples. Data sets with missing values are often "completed" using inference in graphical models [?, ?] or sophisticated low rank matrix factorization techniques [?, ?], which ultimately results in a large, probabilistic database. Data sets that use crowdsourcing are also uncertain [?]. And, very recently, probabilistic databases have been applied to bootstrapping over samples of data [?].

However, probabilistic inference is known to be #P-hard in the size of the database, even for some very simple queries [?]. Today's state of the art inference engines use either sampling based methods or are based on some variant of the DPLL algorithm for Weighted Model Counting. For example Tuffy [?], a popular implementation of Markov Logic Networks (MLN) over relational databases, uses Markov Chain Monte Carlo methods (MCMC). Gibbs sampling can be significantly improved by adapting some classical relational optimization techniques [?]. For another example, MayBMS [?] and its successor Sprout [?] use query plans to guide a DPLL-based algorithm for Weighted Model Counting [?]. While both approaches deploy some advanced relational optimization techniques, at their core they are based on general purpose probabilistic inference techniques, which either run in exponential time (DPLL-based algorithms have been proven recently to take exponential time even for queries computable in polynomial time [?]), or require many iterations until convergence.

In this paper, we propose a different approach to query evaluation on probabilistic databases. In our approach, *every query is evaluated entirely in the database engine*. Probability computation is done at query time, using simple arithmetic operations and aggregates. Thus, probabilistic inference is entirely reduced to a standard query evaluation problem with aggregates. There are no iterations and no exponential blowups. All benefits of relational engines (such as cost-based optimizations, multi-core query processing, shared-nothing parallelization) are immediately available to queries over probabilistic databases. To achieve this, we compute *approximate* rather than exact probabilities, with a one-sided guarantee: The probabilities are guaranteed to be upper bounds to the true probabilities, which we show is *sufficient to rank the top query answers with high precision*. Our approach consists of approximating the true query probability by evaluating a fixed number of "safe queries" (the number depends on the query), each providing an upper bound on the true probability, then taking their minimum.

We briefly review *safe queries*, which are queries whose data complexity is in PTIME. They can be evaluated using *safe query plans* [?, ?, ?], which are related to a technique called *lifted inference* in the AI literature [?, ?]; the entire computation is pushed inside the database engine and is thus efficient. For example, the query $q_1(z) :- R(z,x), S(x,y), K(x,y)$ has the safe query plan $P_1 = \pi_z(R \bowtie_x (\pi_x(S \bowtie_{x,y} K)))$, where every join operator multiplies the probabilities, and every projection with duplicate elimination treats probabilistic events as independent. The literature describes several classes of safe queries [?, ?] and shows that they can be evaluated very efficiently. However, most queries are unsafe: They are provably #P-hard and do not admit safe plans.

In this paper, we prove that every conjunctive query without self-joins can be approximated by a fixed number of safe queries, called "*safe dissociations*" of the original query. Every safe dissociation is guaranteed to return an upper bound on the true probability and can be evaluated in PTIME data complexity. The number of safe dissociations depends only on the query and not the data. Moreover, we show how to find "*minimal safe dissociations*" which are sufficient to find the best approximation to the given query. For example, the unsafe query $q_2(z) :- R(z,x), S(x,y), T(y)$

has two minimal safe dissociations, $q_2'(z) :- R(z,x), S(x,y), T'(x,y)$ and $q_2''(z) :- R'(z,x,y), S(x,y), T(y)$. Both queries are safe and, by setting the probability of every tuple $R'(z,x,y)$ equal to that of $R(z,x)$ and similarly for T', they return an upper bound for the probabilities of each answer tuple from $q_2(z)$. One benefit of our approach is that, if the query happens to be safe, then it has a unique minimal safe dissociation, and our algorithm finds it.

Contributions. (1) We show that there exists a 1-to-1 correspondence between the safe dissociations of a self-join-free conjunctive query and its query plans. One simple consequence is that *every* query plan computes an upper bound of the true probability. For example, the two safe dissociations above correspond to the plans $P_2' = \pi_z(R \Join_x (\pi_x(S \Join_{x,y} T)))$, and $P_2'' = \pi_z((\pi_{zy}(R \Join_x S)) \Join_y T)$. We give an intuitive system R-style algorithm [?] for enumerating all minimal safe dissociations of a query q. Our algorithm takes into account important schema-level information: functional dependencies and whether a relation is deterministic or probabilistic. We prove that our algorithm has several desirable properties that make it a strict generalization of previous algorithms described in the literature: If q is safe then the algorithm returns only one safe plan that computes q exactly; and if q happens to be safe on the particular database instance (e.g., the data happens to satisfy a functional dependency), then one of the minimal safe dissociations will compute the query exactly. (2) We use relational optimization techniques to compute all minimal safe dissociations of a query efficiently in the database engine. Some queries may have a large number of dissociations; e.g., a 8-chain query has 4279 safe dissociations, of which 429 are minimal. Computing 429 queries sequentially in the database engine would still be prohibitively expensive. Instead, we tailor three relational query optimization techniques to dissociation: combining all minimal plans into *one single query*, reusing *common subexpressions* with views, and performing *deterministic semi-join reductions*. (3) We conduct an experimental validation of our technique, showing that, with all our optimizations enabled, computing hard queries over probabilistic databases incurs only a modest penalty over computing the same query on a deterministic database: For example, the 8-chain query runs only a factor of < 10 slower than on a deterministic database. We also show that the dissociation-based technique has high precision for ranking query answers based on their output probabilities.

In summary, our three main contributions are:

(1) We describe an efficient algorithm for finding all minimal safe dissociations for self-join-free conjunctive queries in the presence of schema knowledge. If the query is safe, then our algorithm returns a single minimal plan, which is the safe plan for the query (**??**).

(2) We show how to apply three traditional query optimization techniques to dramatically improve the performance of the dissociation (**??**).

(3) We perform a detailed experimental validation of our approach, showing both its effectiveness in terms of query performance, and the quality of returned rankings. Our experiments also include a novel comparison between deterministic and probabilistic ranking approaches (**??**).

All proofs for this submission together with additional illustrating examples are available in our technical report on arXiv [?].

2. BACKGROUND

Probabilistic Databases. We fix a relational vocabulary $\sigma = (R_1, \ldots, R_m)$. A probabilistic database D is a database plus a function $p(t) \in [0,1]$ associating a probability to each tuple $t \in D$. A *possible world* is a subset of D generated by independently including each tuple t in the world with probability $p(t)$. Thus, the database D is *tuple-independent*. We use bold notation (e.g., $\mathbf{x}$) to denote sets or tuples. A *self-join-free conjunctive query* is a first-order formula $q(\mathbf{y}) = \exists x_1 \ldots \exists x_k.(a_1 \wedge \ldots \wedge a_m)$ where each atom a_i represents a relation $R_i(\mathbf{x}_i)$[1], the variables $x_1, \ldots, x_k$ are called *existential variables*, and $\mathbf{y}$ are called the *head variables* (or free variables). The term "self-join-free" means that the atoms refer to distinct relational symbols. We assume therefore w.l.o.g. that every relational symbol $R_1, \ldots, R_m$ occurs exactly once in the query. Unless otherwise stated, a *query* in this paper denotes a self-join-free conjunctive query. As usual, we abbreviate the query by $q(\mathbf{y}) :- a_1, \ldots, a_m$, and write $\texttt{HVar}(q) = \mathbf{y}$, $\texttt{EVar}(q) = \{x_1, \ldots, x_k\}$ and $\texttt{Var}(q) = \texttt{HVar}(q) \cup \texttt{EVar}(q)$ for the set of head variables, existential variables, and all variables of q. If $\texttt{HVar}(q) = \emptyset$ then q is called a *Boolean* query. We also write $\texttt{Var}(a_i)$ for the variables in atom a_i and $\texttt{at}(x)$ for the set of atoms that contain variable x. The *active domain* of a variable x_i is denoted $ADom_{x_i}$,[2] and the active domain of the entire database is $ADom = \bigcup_i ADom_{x_i}$. The focus of probabilistic query evaluation is to compute $\mathbb{P}(q)$; i.e. the probability that the query is true in a randomly chosen world.

Safe queries, safe plans. It is known that the data complexity of any query q is either in PTIME or #P-hard. The former are called *safe queries* and are characterized precisely by a syntactic property called *hierarchical queries* [?]. We briefly review these results:

DEFINITION 1 (HIERARCHICAL QUERY). *Query q is called hierarchical iff for any $x, y \in \texttt{EVar}(q)$, one of the following three conditions hold: $\texttt{at}(x) \subseteq \texttt{at}(y)$, $\texttt{at}(x) \cap \texttt{at}(y) = \emptyset$, or $\texttt{at}(x) \supseteq \texttt{at}(y)$.*

For example, the query $q_1 :- R(x,y), S(y,z), T(y,z,u)$ is hierarchical, while $q_2 :- R(x,y), S(y,z), T(z,u)$ is not, as neither of the three conditions holds for the variables y and z.

THEOREM 2 (DICHOTOMY [?]). *If q is hierarchical, then $\mathbb{P}(q)$ can be computed in PTIME in the size of D. Otherwise, computing $\mathbb{P}(q)$ is #P-hard in the size of D.*

We next give an equivalent, recursive characterization of hierarchical queries, for which we need a few definitions. We write $\texttt{SVar}(q)$ for the *separator variables* (or root variables); i.e. the set of existential variables that appear in every atom. q is disconnected if its atoms can be partitioned into two non-empty sets that do not share any existential variables (e.g., $q :- R(x,y), S(z,u), T(u,v)$ is disconnected and has two connected components: "$R(x,y)$" and "$S(z,u), T(u,v)$"). For every set of variables $\mathbf{x}$, denote $q - \mathbf{x}$ the query obtained by removing all variables $\mathbf{x}$ (and decreasing the arities of the relation symbols that contain variables from $\mathbf{x}$).

LEMMA 3 (HIERARCHICAL QUERIES). *q is hierarchical iff either: (1) q has a single atom; (2) q has $k \geq 2$ connected components all of which are hierarchical; or (3) q has a separator variable x and $q - x$ is hierarchical.*

DEFINITION 4 (QUERY PLAN). *Let $R_1, \ldots, R_m$ be a relational vocabulary. A query plan P is given by the grammar*

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

where $R_i(\mathbf{x})$ is a relational atom containing the variables $\mathbf{x}$ and constants, $\pi_{\mathbf{x}}$ is the project operator with duplicate elimination*, and $\Join\big[\ldots\big]$ is the* natural join *in prefix notation, 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. $\Join\big[P_1, P_2\big]$ is the same as $\Join\big[P_2, P_1\big]$.*

[1]We assume w.l.o.g. that $\mathbf{x}_i$ is a tuple of only variables without constants.

[2]Defined formally as $ADom_{x_i} = \bigcup_{j: x_i \in \texttt{Var}(R_j)} \pi_{x_i}(R_j)$.

We write $\texttt{HVar}(P)$ for the head variables of P (defined as the variables $\mathbf{x}$ of the top-most projection $\pi_{\mathbf{x}}$ (or the union of the top-most projections if the last operation is a join). Every plan P represents a query q_P defined by taking all atoms mentioned in P and setting $\texttt{HVar}(q_P) = \texttt{HVar}(P)$. For notational convenience, we also use the "project-away" notation, by writing $\pi_{-\mathbf{y}}(P)$ instead of $\pi_{\mathbf{x}}(P)$, where $\mathbf{y}$ are the variables being projected away; i.e. $\mathbf{y} = \texttt{HVar}(P) - \mathbf{x}$.

Given a probabilistic database D and a plan P, each output tuple $t \in P(D)$ has a *score*(t), defined inductively on the structure of P as follows: If $t \in R_i(\mathbf{x})$, then *score*$(t) = p(t)$, i.e. its probability in D; if $t \in \Join\big[P_1(D),\ldots,P_k(D)\big]$ where $t = \Join\big[t_1,\ldots,t_k\big]$, then *score*$(t) = \prod_{i=1}^{k}$ *score*(t_i); and if $t \in \pi_{\mathbf{x}}(P(D))$, and $t_1,\ldots,t_n \in P(D)$ are all the tuples that project into t, then *score*$(t) = 1 - \prod_{i=1}^{n}(1 -$ *score*$(t_i))$. In other words, *score* computes a probability by assuming that all tuples joined by $\Join$ are independent, and all duplicates eliminated by π are also independent. If these conditions hold, then *score* is the correct query probability, but in general the score is different from the probability. Therefore, *score is not equal to the probability, in general*, and is also called an extensional semantics [**?**, **?**]. For a Boolean plan P, we get one single score, which we denote *score*(P).

The requirement that joins and projections alternate is w.l.o.g. because nested joins like $\Join\big[\Join\big[R_1,R_2\big],R_3\big]$ can be rewritten into $\Join\big[R_1,R_2,R_3\big]$ while keeping the same probability score. For the same reason we do not distinguish between different join orders.

DEFINITION 5 (SAFE PLAN). *A plan P is called safe iff, for any join operator $\Join^p\big[P_1,\ldots,P_k\big]$, all subplans have the same head variables:* $\texttt{HVar}(P_i) = \texttt{HVar}(P_j)$ *for all* $1 \le i,j \le k$.

The recursive definition of **??** gives us immediately a safe plan for a hierarchical query. Conversely, every safe plan defines a hierarchical query. The following summarizes our discussion:

PROPOSITION 6 (SAFETY [**?**]). *(1) Let P be a plan for the query q. Then score$(P) = \mathbb{P}(q)$ for any probabilistic database iff P is safe. (2) Assuming #P≠PTIME, a query q is safe (i.e. $\mathbb{P}(q)$ has PTIME data complexity) iff it has a safe plan P; in that case the safe plan is unique, and $\mathbb{P}(q) =$ score(P).*

Boolean Formulas. Consider a set of Boolean variables $\mathbf{X} = \{X_1, X_2, \ldots\}$ and a probability function $p : \mathbf{X} \to [0,1]$. Given a Boolean formula F, denote $\mathbb{P}(F)$ the probability that F is true if each variable X_i is independently true with probability $p(X_i)$. In general, computing $\mathbb{P}(F)$ is #P-hard in the number of variables $\mathbf{X}$. If D is a probabilistic database then we interpret every tuple $t \in D$ as a Boolean variable and denote the lineage of a Boolean $q{:}-g_1,\ldots,g_m$ on D as the Boolean DNF formula $F_{q,D} = \bigvee_{\theta:\theta\models q}\theta(g_1) \wedge \cdots \wedge \theta(g_m)$, where θ ranges over all assignments of $\texttt{EVar}(q)$ that satisfy q on D. It is well known that $\mathbb{P}(q) = \mathbb{P}\big(F_{q,D}\big)$. In other words the probability of a Boolean query is the same as the probability of its lineage formula.

EXAMPLE 7 (LINEAGE). *If $F = XY \vee XZ$ then $\mathbb{P}(F) = p(1-(1-q)(1-r)) = pq + pr - pqr$, where $p = p(X), q = p(Y)$, and $r = p(Z)$. Consider now the query $q{:}-R(x),S(x,y)$ over the database $D = \{R(1),R(2),S(1,4),S(1,5)\}$. Then the lineage formula is $F_{q,D} = R(1) \wedge S(1,4) \vee R(1) \wedge S(1,5)$, i.e. same as F, up to variable renaming. It is now easy to see that $\mathbb{P}(q) = \mathbb{P}\big(F_{q,D}\big)$.*

A key technique that we use in this paper is the following result from [**?**]: Let F, F' be two Boolean formulas with sets of variables $\mathbf{X}$ and $\mathbf{X}'$, respectively. We say that F' is a *dissociation* of F if there exists a substitution $\theta : \mathbf{X}' \to \mathbf{X}$ such that $F'[\theta] = F$. If $\theta^{-1}(X) = \{X', X'', \ldots\}$ then we say that the variable X *dissociates into* $X', X'', \ldots$; if $|\theta^{-1}(X)| = 1$ then we assume w.l.o.g. that $\theta^{-1}(X) = X$ (up to variable renaming) and we say that X does not dissociate. Given a probability function $p : \mathbf{X} \to [0,1]$, we extend it to a probability function $p' : \mathbf{X}' \to [0,1]$ by setting $p'(X') = p(\theta(X'))$. Then, we have shown:

THEOREM 8 (OBLIVIOUS DNF BOUNDS [**?**]). *Let F' be a monotone DNF formula that is a dissociation of F through the substitution θ. Assume that for any variable X, no two distinct dissociations X', X'' of X occur in the same prime implicant of F'. Then (1) $\mathbb{P}(F) \le \mathbb{P}(F')$, and (2) if all dissociated variables $X \in \mathbf{X}$ are deterministic (meaning: $p(X) = 0$ or $p(X) = 1$) then $\mathbb{P}(F) = \mathbb{P}(F')$.*

Intuitively, a dissociation F' is obtained from a formula F by taking different occurrences of a variable X and replacing them with fresh variables $X', X'', \ldots$; in doing this, the probability of F' may be easier to compute, giving us an upper bound for $\mathbb{P}(F)$.

EXAMPLE 9 (**??** CONT.). *$F' = X'Y \vee X''Z$ is a dissociation of $F = XY \vee XZ$, and its probability is $\mathbb{P}(F') = 1-(1-pq)(1-pr) = pq + pr - p^2qr$. Here, only the variable X dissociates into X', X''. It is easy to see that $\mathbb{P}(F) \le \mathbb{P}(F')$. Moreover, if $p = 0$ or 1, then $\mathbb{P}(F) = \mathbb{P}(F')$. The condition that no two dissociations of the same variable occur in a common prime implicant is necessary: for example, $F' = X'X''$ is a dissociation of $F = X$. However $\mathbb{P}(F) = p$, $\mathbb{P}(F') = p^2$, and we do not have $\mathbb{P}(F) \le \mathbb{P}(F')$.*

3. DISSOCIATION OF QUERIES

This section introduces our main technique for approximate query processing. After defining dissociations (**??**), we show that some of them are in 1-to-1 correspondence with query plans, then derive our first algorithm for approximate query processing (**??**). Finally, we describe two extensions in the presence of deterministic relations or functional dependencies (**??**).

3.1 Query dissociation

DEFINITION 10 (DISSOCIATION). *Given a Boolean query $q{:}-R_1(\mathbf{x}_1),\ldots,R_m(\mathbf{x}_m)$ and a probabilistic database D. Let $\Delta = (\mathbf{y}_1,\ldots,\mathbf{y}_m)$ be a collection of sets of variables with $\mathbf{y}_i \subseteq \texttt{Var}(q) - \texttt{Var}(g_i)$ for every relation R_i. The* dissociation *defined by Δ has then two components:*

(1) the dissociated query*: $q^\Delta{:}-R_1^{\mathbf{y}_1}(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m^{\mathbf{y}_m}(\mathbf{x}_m,\mathbf{y}_m)$, where each $R_i^{\mathbf{y}_i}(\mathbf{x}_i,\mathbf{y}_i)$ is a new relation of arity $|\mathbf{x}_i| + |\mathbf{y}_i|$.*

(2) the dissociated database instance *D^Δ consisting of the tables over the vocabulary σ^Δ obtained by evaluating (deterministically) the following queries over the instance D:*

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

where $\mathbf{y}_i = (y_{i1},\ldots,y_{ik_i})$. For each tuple $t' \in R_i^{\mathbf{y}_i}$, its probability is defined as $p'(t') = p(\pi_{\mathbf{x}_i}(t'))$, i.e. the probability of t in the database D.

Thus, a dissociation acts on both the query expression and the database instance: It adds some variables $\mathbf{y}_i$ to each relational symbol R_i of the query expression, and it computes a new instance for each relation $R_i^{\mathbf{y}_i}$ by copying every record $t \in R_i$ once for every tuple in the cartesian product $ADom_{y_{i1}} \times \cdots \times ADom_{y_{ik}}$. When $\mathbf{y}_i = \emptyset$ then we abbreviate $R_i^{\emptyset}$ with R_i. We give a simple example:

EXAMPLE 11 (?? CONT.). *Consider* $q\,{:}{-}\,R(x),S(x,y)$. *Then* $\Delta = (\{y\},\emptyset)$ *defines the following dissociation:* $q^\Delta = R^y(x,y),S(x,y)$*, and the new relation* R^y *contains the tuples* $R^y(1,4),R^y(1,5),R^y(2,4),R^y(2,5)$*. Notice that the lineage of the dissociated query* q^Δ *is* $F_{q^\Delta,D^\Delta} = R^y(1,4),S(1,4) \vee R^y(1,5),S(1,5)$ *and is the same (up to variable renaming) as the dissociation of the lineage of query q:* $F' = X'Y \vee X''Z$*.*

THEOREM 12 (UPPER QUERY BOUNDS). *For every dissociation* Δ *of q:* $\mathbb{P}(q) \leq \mathbb{P}(q^\Delta)$*.*

PROOF. **??** follows immediately from **??** by noting that the lineage F_{q^Δ,D^Δ} is a dissociation of the lineage $F_{q,D}$ through the substitution $\theta : D^\Delta \rightarrow D$ defined as follows: for every tuple $t' \in R_i^{\mathbf{y}_i}$, $\theta(t') = \pi_{\mathbf{x}_i}(t')$. $\square$

DEFINITION 13 (SAFE DISSOCIATION). *A dissociation* Δ *of a query q is called* safe *if the dissociated query* q^Δ *is safe.*

By **??**, a dissociation is safe (i.e. its probability can be evaluated in PTIME) iff q^Δ is hierarchical. Hence, amongst all dissociations, we are interested in those that are easy to evaluate and use them as a technique to approximate the probabilities of queries that are hard to compute. The idea is simple: Find a safe dissociation Δ, compute $\mathbb{P}(q^\Delta)$, and thereby obtain an upper bound on $\mathbb{P}(q)$. In fact, we will consider *all* safe dissociations and take the minimum of their probabilities, since this gives an even better upper bound on $\mathbb{P}(q)$ than that given by a single dissociation. We call this quantity the *propagation score*[3] of the query q.

DEFINITION 14 (PROPAGATION). *The* propagation score $\rho(q)$ for a query q *is the minimum score of all safe dissociations:* $\rho(q) = \min_\Delta \mathbb{P}(q^\Delta)$ *with* Δ *ranging over all safe dissociations.*

The difficulty in computing $\rho(q)$ is that the total number of dissociations is large, even for relatively small queries. If q has k existential variables and m atoms, then q has $2^{|K|}$ possible dissociations with $K = \sum_{i=1}^m \left(k - |\texttt{Var}(g_i)|\right)$ forming a partial order in the shape of a power set lattice (see **??** for **??**). Therefore, our next step is to prune the space of dissociations and examine only the minimum number necessary. We start by defining a partial order on dissociations:

DEFINITION 15 (PARTIAL DISSOCIATION ORDER). *We define the partial order on the dissociations of a query as:*

$$\Delta \preceq \Delta' \;\Leftrightarrow\; \forall i : \mathbf{y}_i \subseteq \mathbf{y}_i'$$

Whenever $\Delta \preceq \Delta'$, then $q^{\Delta'}, D^{\Delta'}$ is a dissociation of q^Δ, D^Δ (given by $\Delta'' = \Delta' - \Delta$). Therefore, we obtain immediately:

COROLLARY 16 (PARTIAL DISSOCIATION ORDER). *If* $\Delta \preceq \Delta'$ *then* $\mathbb{P}(q^\Delta) \leq \mathbb{P}(q^{\Delta'})$*.*

EXAMPLE 17 (PARTIAL DISSOCIATION ORDER). *Consider the query* $q\,{:}{-}\,R(x),S(x),T(x,y),U(y)$*. It is unsafe and allows* $2^3 = 8$ *dissociations which are shown in* **??** *with the help of an "augmented incidence matrix": each row represents one relation and each column one variable: An empty circle (○) indicates that a relation contains a variable; a full circle (●) indicates that a relation is dissociated on a variable (the reason for using two separate symbols becomes clear when we later include domain knowledge).*

[3]The name comes from similarities with efficient belief propagation algorithms in graphical models. See [?] for a discussion on how query dissociation generalizes propagation algorithms from graphs to hypergraphs.

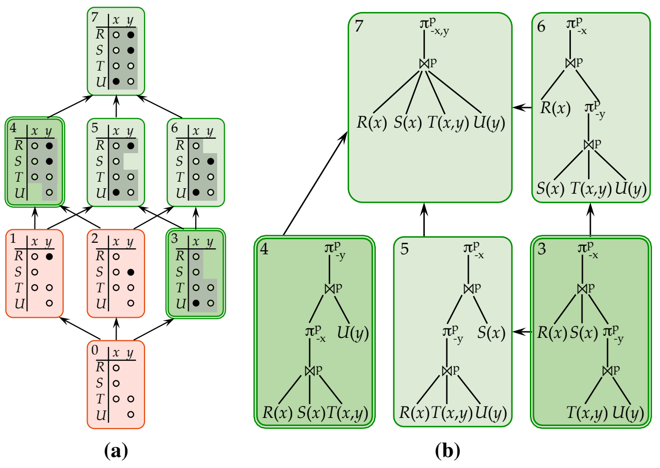


Figure 1: [From Wolfgang: best viewed in color / increase font size in figures]: ?? (a): Partial dissociation order for $q\,{:}{-}\,R(x),S(x),T(x,y),U(y)$**. Safe dissociations are green and have the hierarchy between variables shown (3 to 7),** ***minimal safe dissociations*** **are dark and double-lined (3 and 4). (b): All 5 query plans for** q **and their correspondence to safe dissociations (3 to 7).**

Among those 8 dissociations, 5 are safe, shaded in green, and have the hierarchy among variables highlighted. Furthermore, 2 of the 5 safe dissociations are minimal: $q^{\Delta_3}\,{:}{-}\,R(x),S(x),T(x,y),U^x(x,y)$*, and* $q^{\Delta_4}\,{:}{-}\,R^y(x,y),S^y(x,y),T(x,y),U(y)$ *. To illustrate that these dissociations are upper bounds, consider a database with* $R = T = U = \{1,2\}$*,* $S = \{(1,1),(1,2),(2,2)\}$*, and the probability of all tuples* $= \frac{1}{2}$*. Then q has probability* $\frac{83}{2^9} \approx 0.161$*, while* q^{Δ_3} *has probability* $\frac{169}{2^{10}} \approx 0.165$*, and* q^{Δ_4} *has probability* $\frac{353}{2^{11}} \approx 0.172$*, both of which are upper bounds. The propagation score is the minimum score of all minimal safe dissociations and thus* ≈ 0.165*.* ■

In general, the set of dissociations forms a lattice, with the smallest element $\Delta_\perp = (\emptyset,\ldots,\emptyset)$ ($q^{\Delta_\perp} = q$) and the largest element $\Delta_\top = (\texttt{Var}(q) - \texttt{Var}(g_1),\ldots,\texttt{Var}(q) - \texttt{Var}(g_m))$ ($q^{\Delta_\top}$ is safe, since every atom contains all variables). As we move up in the lattice the probability increases, but the safe/unsafe status may toggle arbitrarily from safe to unsafe and back. For example $q\,{:}{-}\,R(x),S(x),T(y)$ is safe, its dissociation $q'\,{:}{-}\,R(x),S^y(x,y),T(y)$ is unsafe, yet the next dissociation $q''\,{:}{-}\,R(x),S^y(x,y),T^x(x,y)$ is safe again.

This suggests the following naive algorithm for computing $\rho(q)$: Enumerate all dissociations $\Delta_1,\Delta_2,\ldots$ by traversing the lattice breadth-first, bottom up (i.e. whenever $\Delta_i \prec \Delta_j$ then $i < j$). For each dissociation Δ_i, check if q^{Δ_i} is safe. If so, then first update $\rho \leftarrow \min(\rho, \mathbb{P}(q^{\Delta_i}))$, then remove from the list all dissociations $\Delta_j \succ \Delta_i$. However, this algorithm is inefficient for practical purposes for two reasons: (*i*) we need to iterate over many dissociations in order to discover those that are safe; and (*ii*) computing $\mathbb{P}(q^{\Delta_i})$ requires computing a new database instance D^{Δ_i} for each safe dissociation Δ_i. We show in the next section how to avoid both sources of inefficiency by exploiting the lattice structure and by iterating over query plans instead of safe dissociations.

3.2 Dissociations and Plans

We prove here that the safe dissociations q^Δ are in 1-to-1 correspondence with query plans of the original query q. This allows us to (*i*) efficiently find safe dissociations (by iterating over query plans instead of all dissociations), and to (*ii*) compute $\mathbb{P}(q^\Delta)$ without having to materialize the dissociated database D^Δ.

We next describe the 1-to-1 mapping. Consider a safe dissociation q^{Δ} and denote its corresponding unique safe plan P^{Δ}. This plan uses dissociated relations, hence each relation $R_i^{\mathbf{y}_i}(\mathbf{x}_i, \mathbf{y}_j)$ has extraneous variables $\mathbf{y}_i$. Drop all variables $\mathbf{y}_i$ from the relations and all operators using them: This transforms P^{Δ} into a regular, generally unsafe plan P for q. For a trivial example, the plan corresponding to the top dissociation $\Delta_{\top}$ of a query q is $\pi_{-\texttt{Var}(q)}(\bowtie[P_1, \ldots, P_k])$: It performs all joins first, followed by all projections.

Conversely, consider any plan P for q. We define its corresponding safe dissociation Δ^P as follows. For each join operation $\bowtie^p[P_1, \ldots, P_k]$, let its *join variables* $\texttt{JVar}$ be the union of the head variables of all subplans: $\texttt{JVar} = \bigcup_j \texttt{HVar}(P_j)$. For every relation R_i occurring in P_j, add the missing variables $\texttt{JVar} - \texttt{HVar}(P_j)$ to $\mathbf{y}_i$. For example, consider $\bowtie^p[R(x), T(x,y), U(y)]$ (this is the lower join in query plan 5 of **??**). Here, $\texttt{JVar} = \{x, y\}$, and the corresponding safe dissociation of this subplan is $q^{\Delta}(x,y) :- R^y(x,y), T(x,y), U^x(x,y)$. Note that while there is a one-to-one mapping between safe dissociations and query plans, unsafe dissociations do not correspond to plans.

THEOREM 18 (SAFE DISSOCIATION). *Let q be a conjunctive query without self-joins. (1) The mappings $\Delta \mapsto P^{\Delta}$ and $P \mapsto \Delta^P$ are inverses of each other. (2) For every safe dissociation Δ, $\mathbb{P}(q^{\Delta}) = score(P^{\Delta})$.*

COROLLARY 19 (UPPER BOUNDS). *Let P be any plan for a Boolean query q. Then $\mathbb{P}(q) \leq score(P)$.*

The proof follows immediately from $\mathbb{P}(q) \leq \mathbb{P}(q^{\Delta^P})$ (**??**) and $\mathbb{P}(q^{\Delta^P}) = score(P)$ (**??**). In other words, *any* plan for q computes a probability score that is guaranteed to be an upper bound on the correct probability $\mathbb{P}(q)$.

?? suggests the following improved algorithm for computing the propagation score $\rho(q)$ of a query: Iterate over all plans P, compute their scores, and retain the minimum score $\min_P[score(P)]$. Each plan P is evaluated directly on the original probabilistic database, and there no need to materialize the dissociated database instance. However, this approach is still inefficient because it computes several plans that correspond to non-minimal dissociations. For example, in **??** plans 5, 6, 7 correspond to non-minimal dissociations, since plan 3 is safe and below them.

Enumerating minimal safe dissociations. Call a plan P *minimal* if Δ^P is minimal in the set of safe dissociations. For example, in **??**, the minimal plans are 3 and 4. The propagation score is thus the minimum of the scores of the two minimal plans: $\rho(q) = \min_{i \in \{3,4\}}[score(P^{(i)})]$. Our improved algorithm will iterate only over minimal plans, by relying on a connection between plans and sets of variables that disconnect a query: A *cut-set* is a set of existential variables $\mathbf{x} \in \texttt{EVar}(q)$ s.t. $q - \mathbf{x}$ is disconnected. A *min-cut-set* (for minimal cut-set) is a cut-set for which no strict subset is a cut-set. We denote $\texttt{MinCuts}(q)$ the set of all min-cut-sets. Note that q is disconnected iff $\texttt{MinCuts}(q) = \{\emptyset\}$.

The connection between $\texttt{MinCuts}(q)$ and query plans is given by two observations: (1) Let P be any plan for q. If q is connected, then the last operator in P is a projection, i.e. $P = \pi_{-\mathbf{x}}(\bowtie[P_1, \ldots, P_k])$, and the projection variables $\mathbf{x}$ are the join variables $\mathbf{x} = \texttt{JVar}$ because q is Boolean so the plan must project away all variables. We claim that $\mathbf{x}$ is a cut-set for q and that $q - \mathbf{x}$ has k connected components corresponding to $P_1, \ldots, P_k$. Indeed, if P_i, P_j share any common variable y, then they must join on y, hence $y \in \texttt{JVar}$. Thus, *cut-sets are in 1-to-1 correspondence with the top-most projection operator* of a plan. (2) Now suppose that P corresponds to a safe dissociation Δ^P, and let $P' = \pi_{-\mathbf{x}}(\bowtie[P'_1, \ldots, P'_k])$ be its unique safe plan. Then $\mathbf{x} = \texttt{SVar}(q^{\Delta^P})$; i.e. the top-most project operator removes all separator variables.[4] Furthermore, if $\Delta \succeq \Delta^P$ is a larger dissociation, then $\texttt{SVar}(q^{\Delta}) \supseteq \texttt{SVar}(q^{\Delta^P})$ (because any separator variable of a query continues to be a separator variable in any dissociation of that query). Thus, *minimal plans correspond to min-cut-sets*; in other words, $\texttt{MinCuts}(q)$ is in 1-to-1 correspondence with the top-most projection operator of *minimal* plans.

Our discussion leads immediately to **??** for computing the propagation score $\rho(q)$. It also applies to non-Boolean queries by treating the head variables as constants, hence ignoring them when computing connected components. The algorithm proceeds recursively. If q is a single atom then it is safe and we return its unique safe plan. If the query has more than one atom, then we consider two cases, when $q - \texttt{HVar}(q)$ is disconnected or connected. In the first case, every minimal plan is a join, where the subplans are minimal plans of the connected components. In the second case, a minimal plan results from a projection over min-cut-sets. Notice that recursive calls of the algorithm will alternate between these two cases, until they reach a single atom.
[t]

generates all minimal query plans for a given query q.

AlgorithmRecursive algorithm MPMP ForAllforalldoendfch MP (EnumerateMinimalPlans) Query $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \ldots, R_m(\mathbf{x}_m)$ Set of all minimal query plans $\mathscr{P}$ $m = 1$ $\mathscr{P} \leftarrow \{\pi_{\mathbf{x}} R_1(\mathbf{x}_1)\}$ Set $\mathscr{P} \leftarrow \emptyset$ q is disconnected Let $q = q_1, \ldots, q_k$ be the connected components of $q - \texttt{HVar}(q)$ q_iLet $\texttt{HVar}(q_i) \leftarrow \texttt{HVar}(q) \cap \texttt{Var}(q_i)$ $(P_1, \ldots, P_k) \in (q_1) \times \cdots \times (q_k)$ $\mathscr{P} \leftarrow \mathscr{P} \cup \{\bowtie^p[P_1, \ldots, P_k]\}$ $\mathbf{y} \in \texttt{MinCuts}(q - \texttt{HVar}(q))$ Let $q' \leftarrow q$ with $\texttt{HVar}(q') \leftarrow \texttt{HVar}(q) \cup \mathbf{y}$ $P \in (q')$ $\mathscr{P} \leftarrow \mathscr{P} \cup \{\pi_{-\mathbf{y}} P\}$

THEOREM 20 (**??**). **??** *computes the set of all minimal query plans.*

Conservativity. Some probabilistic database systems first check if a query q is safe, and in that case compute the exact probability using the safe plan, otherwise use some approximation technique. We show that **??** is conservative, in the sense that, if q is safe, then $\rho(q) = \mathbb{P}(q)$. Indeed, in that case (q) returns a single plan, namely the safe P for q, because the empty dissociation, $\Delta_{\perp} = (\emptyset, \ldots, \emptyset)$, is safe, and it is the bottom of the dissociation lattice, making it the unique minimal safe dissociation.

Score Quality. We show here that the approximation of $\mathbb{P}(q)$ by $\rho(q)$ becomes tighter as the input probabilities in D decrease. Thus, the smaller the probabilities in the database, the closer does the ranking based on the propagation score approximate the ranking by the actual probabilities.

PROPOSITION 21 (SMALL PROBABILITIES). *Given a query q and database D. Consider the operation of scaling down the probabilities of all tuples in D with a factor $f < 1$. Then the relative error of approximation of $\mathbb{P}(q)$ by the propagation score $\rho(q)$ decreases as f goes to 0:* $\lim_{f \to 0} \frac{\rho(q) - \mathbb{P}(q)}{\mathbb{P}(q)} \to 0$.

Number of Dissociations. While the number of minimal safe dissociations is exponential in the size of the query, recall that it is independent of the size of the database. **??** gives an overview of the number of minimal query plans, total query plans, and all dissociations for k-star and k-chain queries (which are later used in **??**). Later **??** gives optimizations that allow us to evaluate a large number of plans efficiently.

3.3 Minimal plans with schema knowledge

[4]This follows from the recursive definition of the unique safe plan of a query in **??**: the top most projection consists precisely of its separator variables.

k-star query				*k*-chain query			
k	#MP	#P	#Δ	k	#MP	#P	#Δ
1	1	1	1	2	1	1	1
2	2	3	4	3	2	3	4
3	6	13	64	4	5	11	64
4	24	75	4096	5	14	45	4096
5	120	541	$> 10^6$	6	42	197	$> 10^6$
6	720	4683	$> 10^9$	7	132	903	$> 10^9$
7	5040	47293	$> 10^{12}$	8	429	4279	$> 10^{12}$
seq	$k!$	*A000670*	$2^{k(k-1)}$	seq	*A000108*	*A001003*	$2^{(k+1)k}$

Figure 2: Number of minimal plans, total plans, and total dissociations for star and chain queries (A are OEIS sequence numbers [?]).

Next, we show how knowledge of *deterministic relations* (i.e. all tuples have probability = 1), and *functional dependencies* can reduce the number of plans needed to calculate the propagation score.

3.3.1 Deterministic relations (DRs)

Notice that we can treat deterministic relations (DRs) just like probabilistic relations, and **??** with $\mathbb{P}(q) \leq score(P)$ still holds for any plan P. Just as before, our goal is to find a minimum number of plans that compute the minimal score of *all plans*: $\rho(q) = min_P score(P)$. It is known that an unsafe query q can become safe (i.e., $\mathbb{P}(q)$ can be calculated in PTIME with one single plan) if we consider DRs. Thus, in particular, we would still like an improved algorithm that returns one single plan if a query with DRs is safe. The following lemma will help us achieve this goal:

LEMMA 22 (DISSOCIATION AND DRS). *Dissociating a deterministic relation does not change the probability.*

PROOF. **??** follows immediately from Theorem **??** (2) and noting that dissociating tuples in DRs corresponds exactly to dissociating variables $\mathbf{X}$ with $p(X_i) = 1$. □

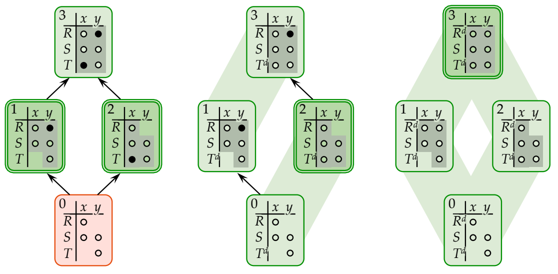
We thus define a new *probabilistic dissociation preorder* $\preceq^p$ by:

$$\Delta \preceq^p \Delta' \Leftrightarrow \forall i, R_i \text{ probabilistic} : \mathbf{y}_i \subseteq \mathbf{y}'_i$$

In other words, $\Delta \preceq^p \Delta'$ still implies $\mathbb{P}(q^\Delta) \leq \mathbb{P}(q^{\Delta'})$, but $\preceq^p$ is defined on probabilistic relations only. Notice, that for queries without DRs, the relations $\preceq^p$ and $\preceq$ coincide. However, for queries with DRs, $\preceq^p$ is a preorder, not an order. Therefore, there exist distinct dissociations Δ, Δ' that are equivalent under $\preceq^p$ (written as $\Delta \equiv^p \Delta'$), and thus have the same probability: $\mathbb{P}(q^\Delta) = \mathbb{P}(q^{\Delta'})$. As a consequence, using $\preceq^p$ instead of $\preceq$, allows us to further reduce the number of minimal safe dissociations.

EXAMPLE 23 (DRS). *Consider $q :- R(x), S(x,y), T^d(y)$ where a d-exponent indicates a DR. This query is known to be safe. We thus expect our definition of $\rho(q)$ to find that $\rho(q) = \mathbb{P}(q)$. Ignore that T^d is deterministic, then $\preceq$ has two minimal plans: $q^{\Delta_1} :- R^y(x,y), S(x,y), T^d(y)$, and $q^{\Delta_2} :- R(x), S(x,y), T^{dx}(x,y)$. Since Δ_2 dissociates only T^d, we now know from* **??** *that $\mathbb{P}(q) = \mathbb{P}\big(q^{\Delta_2}\big)$. Thus, by using $\preceq$ as before, we still get the correct answer. However, evaluating the plan P^{Δ_1} is always unnecessary since $\Delta_2 \preceq^p \Delta_1$. In contrast, without information about DRs, $\Delta_2 \npreceq^p \Delta_1$, and we would thus have to evaluate both plans.*

?? *illustrates this with augmented incidence matrices: dissociated variables in DRs are now marked with empty circles (○) instead of full circles (●), and the preorder $\preceq^p$ is determined entirely by full circles (representing dissociated variables in probabilistic relations). However, as before, the correspondence to plans (as implied by the hierarchy between all variables) is still determined by empty and full circles.* **??** *shows that $\rho(q) = \mathbb{P}\big(q^{\Delta_2}\big) = \mathbb{P}(q)$ since*



(a) **(b) T^d** **(c) R^d and T^d**

Figure 3: $q :- R(x), S(x,y), T(y)$, alternatively with DRs R^d and T^d.

$\Delta_0 \equiv^p \Delta_2 \preceq^p \Delta_1 \equiv \Delta_3$. Thus, the query is safe, and it suffices to evaluate only P^{Δ_2}. Notice that q is not hierarchical, but still safe since it is in an equivalence class with a query that is hierarchical: $\Delta_0 \equiv^p \Delta_2$. **??** *shows that, with R^d and T^d being deterministic, all three possible query plans (corresponding to Δ_1, Δ_2, and Δ_3) form a minimal equivalence class in $\preceq^p$ with Δ^0, and thus give the exact probability. We, therefore, want to modify our algorithm to return just one plan from each minimal safe equivalence class. Ideally, we prefer the plan corresponding to Δ_3 (or more generally, the top plan in $\preceq$ for each minimum equivalence class) since P^{Δ_3} least constrains the join order between tables.*

We now explain two simple modifications to **??** that achieve exactly our desired optimizations described above:

(1) Denote with $\texttt{MinPCuts}(q)$ the set of minimal cut-sets that disconnect the query into at least two connected components with probabilistic tables. Replace $\texttt{MinCuts}(q)$ in **??** with $\texttt{MinPCuts}(q)$.

(2) Denote with m_p the number of probabilistic relations in a query. Replace the stopping condition in **??** with: **if** $m^p \leq 1$ **then** $\mathscr{P} \leftarrow \{\pi_{\mathbf{x}} \bowtie^p \big[R_1(\mathbf{x}_1), \ldots, R_m(\mathbf{x}_m)\big]\}$. In other words, if a query has maximal one probabilistic relation, than join all relations followed by projecting on the head variables.

THEOREM 24 (**??** WITH DRS). **??** *with above 2 modifications returns a minimum number of plans to calculate $\rho(q)$ given schema knowledge about DRs.*

For example, for $q :- R(x), S(x,y), T^d(y)$, $\texttt{MinCuts}(q) = \{\{x\},\{y\}\}$, while $\texttt{MinPCuts}(q) = \{\{x\}\}$. Therefore, the modified algorithm returns P^{Δ_2} as single plan. For $q :- R^d(x), S(x,y), T^d(y)$, the stopping condition is reached (also, $\texttt{MinPCuts}(q) = \{\emptyset\}$) and the algorithm returns P^{Δ_3} as single plan (see **??**).

3.3.2 Functional dependencies (FDs)

Knowledge of functional dependencies (FDs), such as keys, can also restrict the number of necessary minimal plans. A well known example is the query $q :- R(x), S(x,y), T(y)$ from **??**; it becomes safe if we know that S satisfies the FD $\Gamma : x \rightarrow y$ and has a unique safe plan that corresponds to dissociation Δ_2. In other words, we would like our modified algorithm to take Γ into account and to not return the plan corresponding to dissociation Δ_1.

Let Γ be the set of FDs on $\texttt{Var}(q)$ consisting of the union of FDs on every atom R_i in q. As usual, denote $\mathbf{x}_i^+$ the closure of a set of attributes $\mathbf{x}_i$, and denote $\Delta_\Gamma = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$ the dissociation defined as follows: for every atom $R_i(\mathbf{x}_i)$ in q, $\mathbf{y}_i = \mathbf{x}_i^+ \setminus \mathbf{x}_i$. Then we show:

LEMMA 25 (DISSOCIATION AND FDS). *Dissociating a table R_i on any variable $y \in \mathbf{x}_i^+$ does not change the probability.*

This lemma is similar to **??**. We can thus further refine our probabilistic dissociation preorder $\preceq^{p\prime}$ by:

$$\Delta \preceq^{p\prime} \Delta \Leftrightarrow \forall i, R_i \text{ probabilistic} : \mathbf{y}_i \setminus \mathbf{x}_i^+ \subseteq \mathbf{y}_i' \setminus \mathbf{x}_i^+$$

As a consequence, using $\preceq^{p\prime}$ instead of $\preceq^{p}$, allows us to further reduce the number of minimal safe equivalence classes. We next state a result by [**?**] in our notation:

PROPOSITION 26 (SAFETY AND FDS [**?**, PROP. IV.5]). *A query q is safe iff q^{Δ_Γ} is hierarchical.*

This justifies our third modification to **??** for computing $\rho(q)$ of a query q over a database that satisfies Γ: First compute Δ_Γ, then run q^{Δ_Γ} on our previously modified **??**.

THEOREM 27 (**??** WITH FDS). **??** *with above 3 modifications returns a minimum number of plans to calculate $\rho(q)$ given schema knowledge about DRs and FDs.*

It is easy to see that our modified algorithm returns one single plan iff the query is safe, taking into account its structure, DRs and FDs. It is thus a *strict generalization of all known safe self-join-free conjunctive queries* [**?**, **?**]. In particular, we can reformulate the known safe query dichotomy [**?**] in our notation very succinctly:

COROLLARY 28 (DICHOTOMY). *$\mathbb{P}(q)$ can be calculated in PTIME iff there exists a dissociation Δ of q that is (i) hierarchical, and (ii) in an equivalence class with q under $\preceq^{p\prime}$.*

To see what the corollary says, assume first that there are no FDs: Then q is in PTIME iff there exists a dissociation Δ of the DRs only, such that q^{Δ} is hierarchical. If there are FDs, then we first compute the full dissociation Δ_Γ (called "full chase" in [**?**]), then apply the same criterion to q^{Δ_Γ}.

4. MULTI-QUERY OPTIMIZATIONS

So far, **??** enumerates all minimal query plans. We then take the minimum score of those plans in order to calculate the propagation score $\rho(q)$. In this section, we develop three optimizations that can considerably reduce the necessary calculations for evaluating all minimal query plans. Note that these three optimizations and the two optimizations from the previous section are orthogonal and can be arbitrarily combined in the obvious way. We use the following example to illustrate the first two optimizations.

EXAMPLE 29 (OPTIMIZATIONS). *Consider q:– $R(x,z), S(y,u), T(z), U(u), M(x,y,z,u)$. Our default is to evaluate all 6 minimal plans returned by Algorithm **??**, then take the minimum score (shown in Fig. **??**a). Figure **??**b and Fig. **??**c illustrate the optimized evaluations after applying Opt. 1, or Opt. 1 and Opt. 2, respectively.* ■

4.1 Opt. 1: One single query plan

Our first optimization creates one single query plan by *pushing the min-operator down into the leaves*. It thus avoids calculations when it is clear that other calculations must have lower bounds. The idea is simple: Instead of creating one query subplan for each top set $\mathbf{y} \in$ `MinCuts`(q) in **??** of **??**, the adapted **??** takes the minimum score over those top sets, for each tuple of the head variables in **??**. It thus creates one single query plan.

[t]

Optimization 1 recursively pushes the min operator into the leaves and generates one single query plan.

$$\rho(q)=\min\begin{bmatrix}
\boxed{\pi^p_{-z}\bowtie^p[T(z)}, \pi^p_{-x}\bowtie^p[R(x,z),\pi^p_{-u}\bowtie^p[U(u),\boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)}]]]] \\
\boxed{\pi^p_{-z}\bowtie^p[T(z)}, \pi^p_{-u}\bowtie^p[U(u),\pi^p_{-x}\bowtie^p[R(x,z),\boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)}]]]] \\
\boxed{\pi^p_{-z}\bowtie^p[T(z)}, \pi^p_{-u}\bowtie^p[U(u),\pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)]]]] \\
\pi^p_{-u}\bowtie^p[U(u),\pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-z}\bowtie^p[T(z),\pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)]]]] \\
\pi^p_{-u}\bowtie^p[U(u),\pi^p_{-z}\bowtie^p[T(z),\pi^p_{-x}\bowtie^p[R(x,z),\boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)}]]]] \\
\pi^p_{-u}\bowtie^p[U(u),\pi^p_{-z}\bowtie^p[T(z),\pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)]]]]
\end{bmatrix}$$

(a) Result from Algorithm 1: six minimal query plans

$$\rho(q)=\min\begin{bmatrix}
\boxed{\pi^p_{-z}\bowtie^p[T(z)}, \min\begin{bmatrix}\pi^p_{-x}\bowtie^p[R(x,z),\pi^p_{-u}\bowtie^p[U(u),\boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)}]] \\ \pi^p_{-u}\bowtie^p[U(u),\min\begin{bmatrix}\pi^p_{-x}\bowtie^p[R(x,z),\boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)}]] \\ \pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)]]\end{bmatrix}]\end{bmatrix}] \\
\pi^p_{-u}\bowtie^p[U(u),\min\begin{bmatrix}\pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-z}\bowtie^p[T(z),\pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)]]] \\ \pi^p_{-z}\bowtie^p[T(z),\min\begin{bmatrix}\pi^p_{-x}\bowtie^p[R(x,z),\boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)}]] \\ \pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)]]\end{bmatrix}]\end{bmatrix}]
\end{bmatrix}$$

(b) Result from Algorithm 4: one single query plan

$$\begin{aligned}
\underline{V_1(x,z,u)} &= \boxed{\pi^p_{-y}\bowtie^p[S(y,u),M(x,y,z,u)]} \\
V_2(y,z,u) &= \pi^p_{-x}\bowtie^p[R(x,z),M(x,y,z,u)] \\
V_3(z,u) &= \min\begin{bmatrix}\pi^p_{-x}\bowtie^p[R(x,z),\underline{V_1(x,z,u)}] \\ \pi^p_{-y}\bowtie^p[S(y,u),V_2(y,z,u)]\end{bmatrix} \\
\rho(q) &= \min\begin{bmatrix}\boxed{\pi^p_{-z}\bowtie^p[T(z)}\min\begin{bmatrix}\pi^p_{-x}\bowtie^p[R(x,z),\pi^p_{-u}\bowtie^p[U(u),\underline{V_1(x,z,u)}]] \\ \pi^p_{-u}\bowtie^p[U(u),V_3(z,u)]\end{bmatrix}] \\ \pi^p_{-u}\bowtie^p[U(u),\min\begin{bmatrix}\pi^p_{-y}\bowtie^p[S(y,u),\pi^p_{-z}\bowtie^p[T(z),V_2(y,z,u)]] \\ \pi^p_{-z}\bowtie^p[T(z),V_3(z,u)]\end{bmatrix}]\end{bmatrix}
\end{aligned}$$

(c) Result from Algorithm 5: re-using common subplans

Figure 4: ?? before and after applying optimizations 1 and 2.

AlgorithmRecursive algorithm SPSP ForAllforalldoendfch
SP (SinglePlan) Query $q(\mathbf{x})\,{:}{-}\,R_1(\mathbf{x}_1),\ldots,R_m(\mathbf{x}_m)$ Single query plan P

$m=1P \leftarrow \pi^p_{\mathbf{x}}R_i(\mathbf{x}_i)$ q is disconnected Let $q=q_1,\ldots,q_k$ be the components connected by `EVar`(q) Let `HVar`$(q_i) \leftarrow$ `HVar`$(q) \cap$ `Var`(q_i) $P \leftarrow \bowtie^p[(q_1),\ldots,(q_k)]$ Let `MinCuts`$(q)=\{\mathbf{y}_1,\ldots,\mathbf{y}_j\}$ Let $q_i' \leftarrow q_i$ with `HVar`$(q_i') \leftarrow$ `HVar`$(q) \cup \mathbf{y}_i$ **if** $j=1$ **then** $P \leftarrow \pi^p_{-\mathbf{y}_1}(q_1')$ **else** $P \leftarrow \min\left[\pi^p_{-\mathbf{y}_1}(q_1'),\ldots,\pi^p_{-\mathbf{y}_j}(q_j')\right]$

4.2 Opt. 2: Re-using common subplans

Our second optimization calculates only once, then *re-uses common subplans shared between the minimal plans*. Thus, whereas our first optimization reduces computation by combining plans at their roots, the second optimization stores and re-uses common results in the branches. The adapted **??** works as follows: It first traverses the whole single query plan (FindingCommonSubplans) and remembers each subplan by the atoms used and its head variables in a HashSet HS (**??**). If it sees a subplan twice (**??**), it creates a new view for this subplan, mapping the subplan to a new view definition. The actual plan (ViewReusingPlan) then uses these views whenever possible (**??**). The order in which the views are created (**??**) assures that the algorithm also discovers and exploits *nested common subexpressions*. Figure **??**c illustrates for **??**, that both the main plan and the view V_3 re-use views V_1 and V_2.
[t]

Optimizations 1 & 2 together create a query plan which re-uses several previously defined temporary views.

AlgorithmAlgorithm FunctionRecursive function FSFS SPSP RPRP HSHS HMHM ForAllforalldoendfch

UsingCommonSubplans Query $q(\mathbf{x})\,{:}{-}\,R_1(\mathbf{x}_1),\ldots,R_m(\mathbf{x}_m)$ Ordered set of view definitions $\mathscr{V}$, final query plan $P \leftarrow \emptyset$ // HashSet of all subplans $\leftarrow(\emptyset,\emptyset)$ // HashMap from subplans to unique view names $\mathscr{V} \leftarrow \emptyset$ // Set of view definitions (q) $q_i \in$.keys in increasing size of `HVar`(q_i) and `Var`(q_i) $\mathscr{V} \leftarrow \mathscr{V} \cup \{.val = \mathit{ViewReusingPlan}(q_i)\}$ $P=(q)$

FS (FindingCommonSubplans) Query $q(\mathbf{x})\,{:}{-}\,R_1(\mathbf{x}_1),\ldots,R_m(\mathbf{x}_m)$ q is disconnected Let $q=q_1,\ldots,q_k$ be the components connected by `EVar`(q) $q_i(q_i(\mathbf{x}_i))$ $(m=1 \wedge \mathbf{x}=\mathbf{x}_i) \vee (q) \neq \emptyset$ $q \in \wedge(q)=\emptyset$ $(q) \leftarrow$ new view name $HS \leftarrow HS \cup \{q\}$

$\mathbf{y} \in \texttt{MinCuts}(q)$ Let $q' \leftarrow q$ with $\texttt{HVar}(q') \leftarrow \texttt{HVar}(q) \cup \mathbf{y}$ (q') RP (ViewReusing-Plan) Query $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \ldots, R_m(\mathbf{x}_m)$ Query plan P that reuses views from HashMap HM **if** $(q) \neq \emptyset$ **then** $P \leftarrow (q)$ *Insert here lines 1-11 from* **??**, *replacing with*

4.3 Opt. 3: Deterministic semi-join reduction

The most expensive operations in probabilistic query plans are the group-bys for the probabilistic project operations. These are often applied early in the plans to tuples which are later pruned and do not contribute to the final query result. Our third optimization is to first apply a *full semi-join reduction on the input relations* before starting the probabilistic evaluation from these *reduced input relations*. We like to draw here an important connection to [**?**], which introduces the idea of "lazy plans" and shows orders of magnitude performance improvements for safe plans by computing confidences not after each join and projection, but rather at the very end of the plan. We note that our semi-join reduction *serves the same purpose* with similar performance improvements and also apply for safe queries. The advantage of semi-join reductions, however, is that we do not require any modifications to the query engine.

5. EXPERIMENTS

We are interested in both the quality and the efficiency of dissociation as compared to exact probabilistic inference, Monte Carlo simulation (MC), and standard deterministic query evaluation ("deterministic SQL"). Our experiments, thus, investigate the following questions: *How much can our three optimizations improve dissociation? How fast is dissociation as compared to exact probabilistic inference, MC, and deterministic query evaluation? How good is the ranking from dissociation as compared to MC and ranking by lineage size? What are the most important parameters determining the ranking quality for each of the three methods?*

Ranking quality. We use *mean average precision* (MAP) to evaluate the quality of ranking of a method by comparing it against the ranking from exact probabilistic inference as ground truth (GT). MAP rewards rankings that place relevant items earlier; the best possible value is 1, and the worst possible 0 [**?**]. Average Precision at 10 (AP@10) is defined as $\text{AP@10} = \frac{\sum_{k=1}^{10} \text{P@}k}{n}$, where P@$k$ is the precision at kth answer returned. Averaging over several rankings yields MAP. We use a variant of the analytic method proposed in [**?**] to calculate AP in the presence of ties. As baseline for no ranking, we assume all tuples to have the same score and thus be tied for the same position and call this *random average precision*.

Exact probabilistic inference. Whenever possible, we calculate GT rankings with a tool called SampleSearch [**?**, **?**], which also serves to evaluate the cost of exact probabilistic inference. We describe the method of transforming the lineage DNF into a format that can be read by SampleSearch in [**?**].

Monte Carlo (MC). We evaluate the MC simulations for different numbers of samples and write MC(x) for x samples. For example, AP for MC(10k) is the result of sampling the individual tuple scores 10 000 times from their lineages and then evaluating AP once over the sampled scores. The MAP scores together with the standard deviations are then the average over several repetitions.

Ranking by lineage size. To evaluate the potential of non-probabilistic methods for ranking answers, we also rank the answer tuples by decreasing size of their lineages; i.e. number of terms. Intuitively, a larger lineage size should indicate that an answer tuple has more "support" and should thus be more important.

Setup 1. We use the TPC-H DBGEN data generator [**?**] to generate a 1GB database to which we add a column P for each table and store it in PostgreSQL 9.2 [**?**]. We assign to each input tuple i a random probability p_i uniformly chosen from the interval $[0, p_{i\max}]$, resulting in an expected average input probability $\text{avg}[p_i] = p_{i\max}/2$. By using databases with $\text{avg}[p_i] < 0.5$, we can avoid output probabilities close to 1 for queries with very large lineages. We use the following parameterized query:

$$Q(a) :- S(\underline{s}, a), PS(s, u), P(\underline{u}, n), s \leq \$1, n \text{ like } \$2$$

```
select distinct s_nationkey from Supplier, Partsupp, Part
where s_suppkey = ps_suppkey and ps_partkey = p_partkey
and s_suppkey <= $1 and p_name like $2
```

Parameters \$1 and \$2 allow us to change the lineage size. Tables Supplier, Partsupp and Part have 10k, 800k and 200k tuples, respectively. There are 25 different numeric attributes for nationkey and our goal is to efficiently rank these 25 nations. As baseline for not ranking, we use random average precision for 25 answers, which leads to MAP@10 = 0.220. This query has two minimal query plans and we will compare the speed-up from either evaluating both individually or performing a deterministic semi-join reduction (Optimization 3) on the input tables.

Setup 2. We compare the run times for our three optimizations against evaluation of all plans for k-chain queries and k-star queries over varying database sizes (data complexities) and varying query sizes (query complexities). The k-chain queries have arity $= 2$ and several results, whereas the star queries have arity $= 1$ and cardinality $= 1$, representing a Boolean query:

$$k\text{-chain: } q(x_0, x_k) :- R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_k(x_{k-1}, x_k)$$

$$k\text{-star: } q('a') :- R_1('a', x_1), R_2(x_2), \ldots, R_k(x_k), R_0(x_1, \ldots, x_k)$$

We denote the length of the query with k, the number of tuples per table with n, and the domain size with N. We use integer values which are uniformly randomly drawn from the range $\{0, 1, \ldots N-1\}$. This parameter determines the *selectivity* and is varied as to keep the *answer cardinality constant* around 20-50 for chain queries, and the answer probability between 0.90 and 0.95 for star queries. For the data complexity experiments, we vary the number of tuples n per table between 100 and 10^6. For the query complexity experiments, we vary k between 2 and 8 for chain queries. For these experiments, the optimized (and often *extremely long*) SQL statements are "calculated" in JAVA and then sent to Microsoft SQL server 2012.

5.1 Run time experiments

QUESTION 1. *When and how much do our three query optimizations speed up query evaluation?*

Result 1. Combining plans (Opt. 1) and using intermediate views (Opt. 2) almost always speeds up query times. The semi-join reduction (Opt. 3) slows down queries with high selectivities, but considerably speeds up queries with small selectivities.

Figures **??** to **??** show the results on setup 2 for increasing database sizes or query sizes. For example, **??** shows the performance of computing a 7-chain query which has 132 safe dissociations. Evaluating each of these queries separately takes a long time, while our optimization techniques bring evaluation time close to deterministic query evaluation. Especially on larger databases, where the running time is I/O bound, the penalty of the probabilistic inference is only a factor of 2-3 in this example. Notice here the trade-off between optimization 1,2 and optimization 1,2,3: Optimization 3 applies a full semi-join reduction on the input relations before starting the probabilistic plan evaluation from these reduced input relations. This operation imposes a rather large constant overhead, both at the query optimizer and at query execution. For larger databases (but constant selectivity), this overhead is amortized. In practice, this suggests that dissociation allows us a large space of optimizations depending on the query and particular database instance that can conservatively extend the space of optimizations performed today in deterministic query optimizers.

Figures **??** to **??** compare the running times on setup 1 between dissociation with two minimal query plans ("Diss"), dissociation with semi-join reduction ("Diss + Opt3"), exact probabilistic inference ("SampleSearch"), Monte Carlo with 1000 samples ("MC(1k)"), retrieving the lineage only ("Lineage query"), and deterministic query evaluation without ranking ("Standard SQL"). We fixed $\$2 \in \{\text{'\%red\%green\%'}, \text{'\%red\%'}, \text{'\%'}\}$ and varied $\$1 \in \{500, 1000, \ldots 10k\}$. **??** combines all three previous plots and shows the times as function of the maximum lineage size (i.e. the size of the lineage for the tuple with the maximum lineage) of a query. We see here again that the semi-join reduction speeds up evaluation considerably for small lineage sizes (**??** shows speedups of up to 36). For large lineages, however, the semi-join reduction is an unnecessary overhead, as most tuples are participating in the join anyway (**??** shows overhead of up to 2).

QUESTION 2. *How does dissociation compare against other probabilistic methods and standard query evaluation?*

Result 2. The best evaluation strategy for dissociation takes only a small overhead over standard SQL evaluation and is considerably faster than other probabilistic methods for large lineages.

Figures **??** to **??** show that SampleSearch does not scale to larger lineages as the performance of exact probabilistic inference depends on the tree-width of the Boolean lineage formula, which generally increases with the size of the data. In contrast, dissociation is *independent of the treewidth*. For example, SampleSearch needed 780 sec for calculating the ground truth for a query with $\max[\text{lin}] = 5.9\text{k}$ for which dissociation took 3.0 sec, and MC(1k) took 42 sec for a query with $\max[\text{lin}] = 4.2\text{k}$ for which dissociation took 2.4 sec. Dissociation takes only 10.5 sec for our largest query $\$2 = \text{'\%'}$ and $\$1 = 10k$ with $\max[\text{lin}] = 35\text{k}$. Retrieving the lineage for that query alone takes 5.8 sec, which implies that any probabilistic method that evaluates the probabilities outside of the database engine needs to issue this query to retrieve the DNF for each answer and would thus have to evaluate lineages of sizes around 35k in only 4.7 (= 10.5 - 5.8) sec to be faster than dissociation.[5]

5.2 Ranking experiments

For the following experiments, we are limited to those query parameters \$1 and \$2 for which we can get the ground truth (and results from MC) in acceptable time. We systematically vary $p_{i\max}$ between 0.1 and 1 (and thus $\text{avg}[p_i]$ between 0.05 and 0.5) and evaluate the rankings several times over randomly assigned input tuple probabilities. We only keep data points (i.e. results of individual ranking experiments) for which the output probabilities are not too close to 1 to be meaningful ($\max[p_a] < 0.999999$).

QUESTION 3. *How does ranking quality compare for our three ranking methods and which are the most important factors that determine the quality for each method?*

Result 3. Dissociation performs better than MC which performs better than ranking by lineage size.

[5]The time needed for the lineage query thus serves as minimum benchmark for *any* probabilistic approximation. The reported times for SampleSearch and MC are the sum of time for retrieving the lineage plus the actual calculations, without the time for reading and writing the input and output files for SampleSearch.

?? shows averaged results of our probabilistic methods for $\$2 = \text{'\%red\%green\%'}$.[6] Shaded areas indicate standard deviations and the x-axis shows varying numbers of MC samples. We only used those data points for which $\text{avg}[p_a]$ of the top 10 ranked tuples is between 0.1 and 0.9 according to ground truth ($\approx$ 6k data points for dissociation and lineage, $\approx$ 60k data points for MC, as we repeated each MC simulation 10 times), as this is the best regime for MC, according to **??**. We also evaluated quality for dissociation and ranking by lineage for more queries by choosing parameter values for \$2 from a set of 28 strings, such as '%r%g%r%a%n%d%' and '%re%re%'. The average MAP over all 28 choices for parameters \$2 is 0.997 for ranking by dissociation and 0.520 for ranking by lineage size ($\approx$ 100k data points). Most of those queries have too large of a lineage to evaluate MC. Note that ranking by lineage always returns the same ranking for given parameters \$1 and \$2, but the GT ranking would change with different input probabilities.

Result 4. Ranking quality of MC increases with the number of samples and decreases when the average probability of the answer tuples $\text{avg}[p_a]$ is close to 0 or 1.

?? shows the AP as a function of $\text{avg}[p_a]$ of the top 10 ranked tuples according to ground truth by logarithmic scaling of the x-axis (each point in the plot averages AP over $\approx$ 450 experiments for dissociation and lineage and over $\approx$ 4.5k experiments for MC). We see that MC performs increasingly poor for ranking answer tuples with probabilities close to 0 or 1 and even approach the quality of random ranking (MAP@10 = 0.22). This is so because, for these parameters, the probabilities of the top 10 answers are very close, and MC needs many iterations to distinguish them. Therefore, MC performs increasingly poorly for increasing size of lineage but fixed average input probability $\text{avg}[p_i] \approx 0.5$, as the average answer probabilities $\text{avg}[p_a]$ will be close to 1. In order not to "bias against our competitor," we compared against MC in its best regime with $0.1 < \text{avg}[p_a] < 0.9$ in **??**.

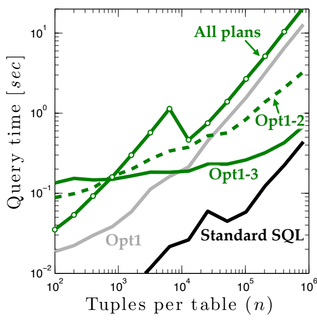
Result 5. Ranking by lineage size has good quality only when all input tuples have the same probability.

?? shows that ranking by lineage is good only when all tuples in the database have the *same* probability (labeled by $p_i = \text{const}$ as compared to $\text{avg}[p_i] = \text{const}$). This is a consequence of the output probabilities depending mostly on the size of the lineages if all probabilities are equal. Dependence on other parameters, such as overall lineage size and magnitude of input probabilities (here shown for $p_i = 0.1$ and $p_i = 0.5$), seem to matter only slightly.

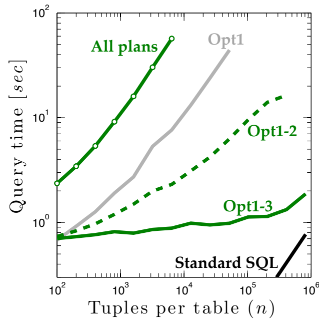
Result 6. The quality of dissociation decreases with the average number of dissociations per tuple $\text{avg}[d]$ and with the average input probabilities $\text{avg}[p_i]$. Dissociation performs very well and notably better then MC(10k) if either $\text{avg}[d]$ or $\text{avg}[p_i]$ are small.

Each answer tuple a gets its score p_a from one of two query plans P_S and P_P that dissociate tuples in tables S and P, respectively. For example, if the lineage size for tuple a is 100 and the lineage contains 20 unique suppliers from table S and 50 unique parts from table P, then P_S dissociates each tuple from S into 5 tuples and P_P each tuple from P into 2 tuples, on average. Most often, P_P will then give the better bounds as it has fewer average dissociations. Let $\text{avg}[d]$ be the mean number of dissociations for each tuple in the dissociated table of its respective optimal query plan, averaged

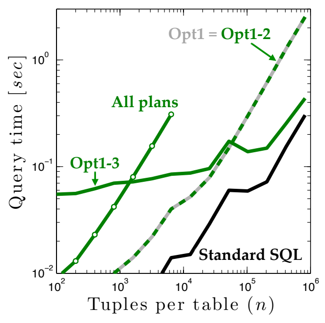
[6]Results for MC and other parameters of \$2 are similar. However, the evaluation time for the experiments becomes quickly infeasible.



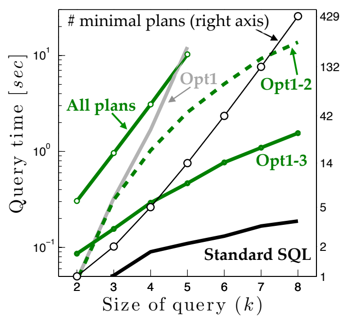
(a) 4-chain queries



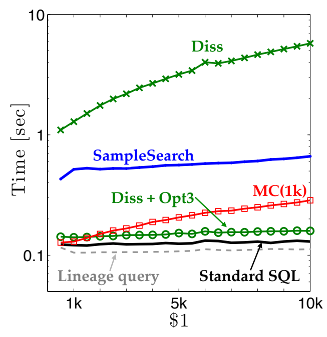
(b) 7-chain queries



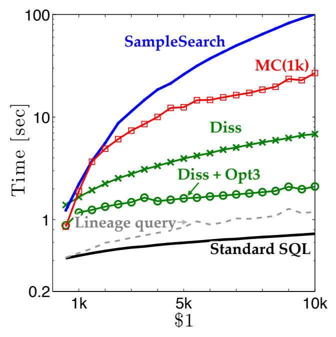
(c) 2-star queries



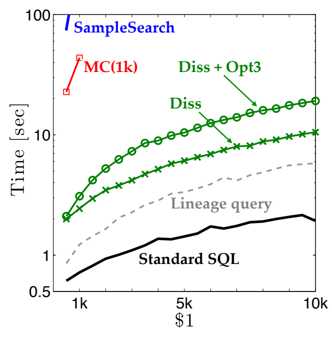
(d) k-chain queries



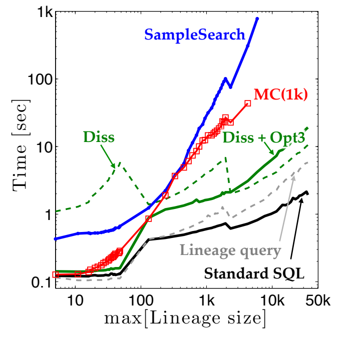
(e) $2 = %red%green%



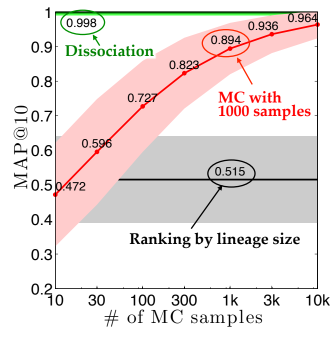
(f) $2 = %red%



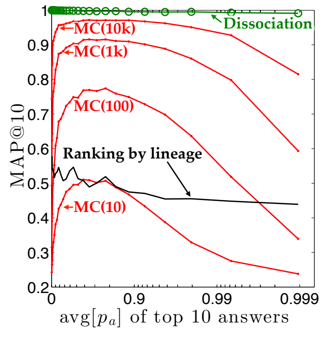
(g) $2 = %



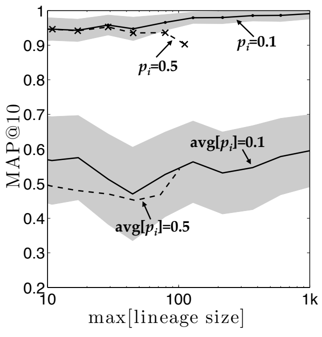
(h) Combining (a)-(c)



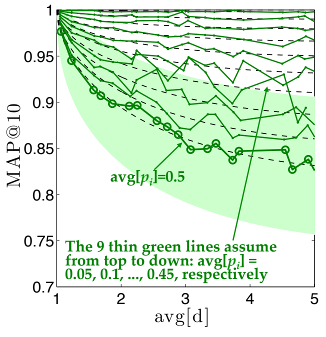
(i) ??



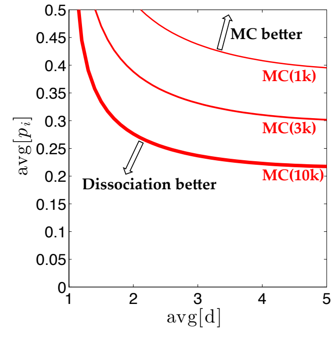
(j) ??



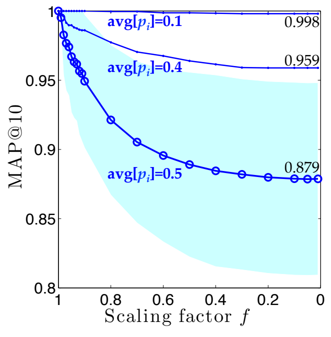
(k) ??



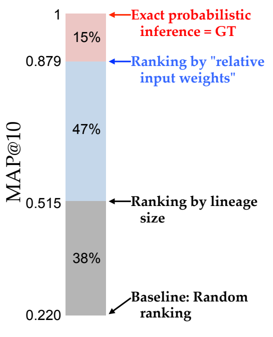
(l) ??



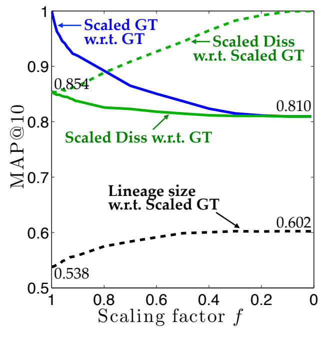
(m) ??



(n) ??



(o) ??



(p) ??

Figure 5: Timing results: (a)-(c) For increasing database sizes and constant cardinalities, our optimizations approach deterministic SQL performance. (d) Our optimizations can even evaluate very large number of minimal plans efficiently (here shown up to 429 for an 8-chain query). (e)-(h) For the TPC-H query, the best evaluation for dissociation is within a factor of 6 of that for deterministic query evaluation. (i)-(p) Ranking experiments on TPC-H: Assumptions for each subfigure and conclusions that can be drawn are described in the main text in the respective result paragraph. **[From Wolfgang: increase font size for annotations in figure!!!]**

across all top 10 ranked answer tuples. For all our queries (even those with \$1 = $10k$ and \$2 = '%'), avg$[d]$ stays below 1.1 as, for each tuple, there is usually one plan that dissociates few variables. In order to *understand the impact of higher numbers of dissociations* (increasing avg$[d]$), we also measured AP for the ranking for *each query plan individually*. Hence, for each choice of random parameters, we record two new data points – one for ranking all answer tuples by using only P_S and one for using only P_P – together with the values of avg$[d]$ in the respective table that gets dissociated. This allows us to draw conclusions for a larger set of parameters. **??** plots MAP values as a function of avg$[d]$ of the top 10 ranked tuples on the horizontal axis, and various values of avg$[p_i]$ (avg$[p_i] = 0.05, 0.10, \ldots, 0.5$). Each plotted point averages over at least 10 data points (some have 10, other several 1000s). Dashed lines show a fitted parameterized curve to the data points on avg$[p_i]$ and avg$[d]$. The figure also shows the standard deviations as shaded areas for avg$[p_i] = 0.5$. We see that the quality is very dependent on avg$[p_i]$, as predicted by **??**.

?? maps the trade-off between dissociation and MC for the two important parameters for the quality of dissociation (avg$[d]$ and avg$[p_i]$) and the number of samples for MC. For example, MC(1k) gives a better expected ranking than dissociation only for the small area above the thick red curve marked MC(1k). For MC, we used the test results from **??**; i.e. assuming $0.1 < \text{avg}[p_a] < 0.9$ for MC. Also recall that for large lineages, having an input probability with avg$[p_i] = 0.5$ will often lead to answer probabilities close to 1 for which ranking is not possible anymore (recall **??**). Thus, for large lineages, we need small input probabilities to have meaningful interpretations. And for small input probabilities, dissociation considerably outperforms any other method.

QUESTION 4. *How much would the ranking change according to exact probabilistic inference if we scale down all input tuples?*

Result 7. If the probabilities of all input tuples are already small, then scaling them further down does not affect the ranking much.

Here, we repeatedly evaluated the exact ranking for 7 different parameterized queries over randomly generated databases with one query plan that has avg$[d] \approx 3$, for two conditions: first on a probabilistic database with $avg[p_i]$ input probabilities (we defined the resulting ranking as GT); then again on a scaled version, where all input probabilities in the database are multiplied by the same scaling factor $f \in (0, 1)$. We then compared the new ranking against GT. **??** shows that if all input probabilities are already small (and dissociation already works well), then scaling has little effect on the ranking. However, for $avg[p_i] = 0.5$ (and thus many tuples with p_i close to 1), we have a few tuples with p_i close to 1. These tuples are very influential for the final ranking, but their relative influence decreases if scaled down even slightly. Also note that even for $avg[p_i] = 0.5$, scaling a database by a factor $f = 0.01$ instead of $f = 0.2$ does not make a big difference. However, the quality remains well above ranking by lineage size (!). This suggests that the difference between ranking by lineage size (MAP = 0.529) and the ranking on a scaled database for $f \to 0$ (MAP = 0.879) can be attributed to the relative weights of the input tuples (we thus refer to this as "*ranking by relative input weights*"). The remaining difference in quality then comes from the *actual probabilities* assigned to each tuple. Using MAP = 0.220 as baseline for random ranking, 38% of the ranking quality can be found by the lineage size alone vs. 85% by the lineage size plus the relative weights of input tuples. The remaining 15% come from the actual probabilities (**??**).

QUESTION 5. *Does the expected ranking quality of dissociation decrease to random ranking for increasing fractions of dissociation (just like MC does for decreasing number of samples)?*

Result 8. The expected performance of dissociation for increasing avg$[d]$ for a particular query is lower bounded by the quality of ranking by relative input weights.

Here, we use a similar setup as before and now compare various rankings against each other: SampleSearch on the original database ("GT"); SampleSearch on the scaled database ("Scaled GT"); dissociation on the scaled database ("Scaled Diss"); and ranking by lineage size (which is unaffected by scaling). From **??**, we see that the quality of Scaled Diss w.r.t. Scaled GT $\to 1$ for $f \to 0$ since dissociation works increasingly well for small avg$[p_i]$ (recall **??**). We also see that Scaled Diss w.r.t. GT decreases towards Scaled GT w.r.t. GT for $f \to 0$. Since dissociation can always reproduce the ranking quality of ranking by relative input weights by first downscaling the database (though losing information about the actual probabilities) the expected quality of dissociation for smaller scales does not decrease to random ranking, but rather to ranking by relative weights. Note this result only holds for the expected MAP; any particular ranking can still be very much off.

6. RELATED WORK

Probabilistic databases. Current approaches to query evaluation on probabilistic databases can be classified into three categories: (*i*) *incomplete approaches* identify tractable cases either at the query-level [**?**, **?**, **?**] or the data-level [**?**, **?**, **?**]; (*ii*) *exact approaches* [**?**, **?**, **?**, **?**, **?**] work well on queries with simple lineage expressions, but perform poorly on database instances with complex lineage expressions. (*iii*) *approximate approaches* either apply general purpose sampling methods [**?**, **?**, **?**, **?**], or approximate the number of models of the Boolean lineage expression [**?**, **?**, **?**]. Our work can be seen as a generalization of several of these techniques: Our algorithm returns the exact score if the query is safe [**?**, **?**] or data-safe [**?**].

Lifted and approximate inference. Lifted inference was introduced in the AI literature as an approach to probabilistic inference that uses the first-order formula to exploit symmetries at the grounded level [**?**]. This research evolved independently of that on probabilistic databases, and the two have many analogies: A formula is called *domain liftable* iff its data complexity is in polynomial time [**?**], which is the same as a *safe query* in probabilistic databases, and the FO-d-DNNF circuits described in [**?**] correspond to the safe plans discussed in this paper. See [**?**] for a recent discussion on the similarities and differences.

Representing Correlations. The most popular approach to represent correlations between tuples in a probabilistic database is by a Markov Logic network (MLN) which is a set of *soft constraints* [**?**]. Quite remarkably, all complex correlations introduced by an MLN can be rewritten into a query over a tuple-independent probabilistic database [**?**, **?**, **?**]. In combination with such rewritings, our techniques can be also applied to MLNs if their rewritings results in conjunctive queries without self-joins.

Dissociation. Dissociation was first introduced in the workshop paper [**?**], presented as a way to generalize graph propagation algorithms to hypergraphs. Theoretical upper and lower bounds for dissociation of Boolean formulas, including **??**, were proven in [**?**]. Dissociation is related to a technique called *relaxation* for probabilistic inference in graphical models [**?**].

7. CONCLUSIONS AND OUTLOOK

This paper proposes to approximate probabilistic query evaluation by evaluating a fixed number of safe queries, each providing an upper bound on the true probability, then taking their minimum. We provide an algorithm that takes into account important schema information to enumerate only the minimal necessary plans among all possible plans, and prove it to be a strict generalization of all known results of PTIME self-join free conjunctive queries. We describe relational query optimization techniques that allow us to evaluate all minimal queries in a single query and very fast. Our evaluations show that the optimizations of our approach bring probabilistic query evaluation close to standard query evaluation while providing high ranking quality. In future work, we plan to generalize the approach to full first-order queries.

Acknowledgements. This work was supported in part by NSF grants IIS-0513877, IIS-0713576, IIS-0915054, and IIS-1115188. We thank the reviewers for their careful reading of this manuscript and their detailed feedback. WG would also like to thank Manfred Hauswirth for a question in 2007 that was crucial for this paper.