

Dissociation and propagation for approximate lifted inference with standard relational database management systems

Wolfgang Gatterbauer¹ · Dan Suciu²

Received: 31 December 2015 / Revised: 25 April 2016 / Accepted: 13 June 2016 / Published online: 16 July 2016
© Springer-Verlag Berlin Heidelberg 2016

Abstract Probabilistic inference over large data sets is a challenging data management problem since exact inference is generally #P-hard and is most often solved approximately with sampling-based methods today. This paper proposes an alternative approach for approximate evaluation of conjunctive queries with standard relational 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 PTIME self-join-free conjunctive queries*: A query is in PTIME if and only if our algorithm returns one single plan. Furthermore, our approach is a generalization of a family of efficient ranking methods from graphs to hypergraphs. We also adapt three relational query optimization techniques to evaluate all necessary 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. We also note that the techniques developed in this paper apply immediately to *lifted inference* from sta-

tistical relational models since lifted inference corresponds to PTIME plans in probabilistic databases.

Keywords Probabilistic inference · Lifted inference · Probabilistic databases · Problem relaxation · Ranking · Query plans · Query optimization

1 Introduction

Probabilistic inference over large data sets is becoming a central data management problem. Recent large knowledge bases, such as Yago [39], Nell [7], DeepDive [15], or Google’s Knowledge Vault [18], have millions to billions of uncertain tuples. Data sets with missing values are often “completed” using inference in graphical models [8,60,70] or sophisticated low rank matrix factorization techniques [20,69] that ultimately result in a large probabilistic database. Data sets that result from crowdsourcing [1] or that are inferred from unstructured information [9] are also uncertain, and probabilistic databases have been applied to bootstrapping over samples of data [79].

However, probabilistic inference is known to be #P-hard in the size of the database, even for some very simple queries [12]. 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 [14]. For example, Tuffy [50], 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 [80]. For another example, MayBMS [3] and its successor Sprout [54] use query plans to guide a DPLL-based algorithm for Weighted Model Counting [34]. While both approaches

Electronic supplementary material The online version of this article (doi:10.1007/s00778-016-0434-5) contains supplementary material, which is available to authorized users.

✉ Wolfgang Gatterbauer
gatt@cmu.edu

¹ Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

² Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

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 [4]), or require many iterations until convergence.

In this paper, we propose a different approach to query evaluation with probabilistic databases (PDBs). 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 directly 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 #P-hard query probability by evaluating a fixed number of PTIME queries (the number depends on the query), each providing an upper bound on the true probability, then taking their minimum. Another way to put this is that we replace the standard semantics based on reliability, with a related but much more efficient semantics based on propagation, and which is guaranteed to be an upper bound on reliability. We explain this alternative semantics next.

The semantics of a query over a PDB is based on the possible world semantics, which is equivalent to “query reliability” [36]. Among its roots are network reliability [10] which is defined as the probability that a source node s remains connected to a target node t in a directed graph if edges fail independently with known probabilities. However, computing network reliability is #P-hard. Hence, many applications where an exact probabilistic semantics is not critical (especially for ranking alternative answers) have replaced network reliability with another semantics 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. Intuitively, let’s call a node x “active” if there exists a directed path from the source node to x . Then, the “reliability score” $r(x)$ of a node x is the probability that x is active if every edge e is included in the graph independently with probability p_e . The score of interest is the reliability of a target node t : $r(t) = p_1(p_2p_4 \otimes p_3p_5) = p_1(1 - (1 - p_2p_4)(1 - p_3p_5))$ where “ \otimes ” stands for the “independent-or” in infix or prefix notation, which combines probabilities as if calculating the disjunction between independent events: $\otimes_i p_i := 1 - \prod_i (1 - p_i)$. While reliability

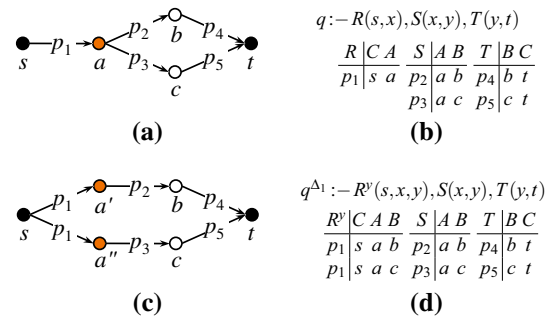


Fig. 1 Example 1. The propagation score $\rho(t)$ in graph a corresponds to the reliability score $r(t)$ in graph c with node a dissociated into two. b, d Corresponding chain queries with respective databases

can be computed efficiently for series-parallel graphs as the one in Fig. 1a, it is #P-hard in general, even on 4-partite networks [10]. 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 3-chain query $q: -R(s, x), S(x, y), T(y, t)$ over the PDB shown in Fig. 1b (here s and t stand for constants). Notice that the reliability of a node is a combinatorial or “global property” of the entire graph: it is defined as a weighted average over all possible worlds and can generally not be calculated easily.

In contrast, the propagation score $\rho(x)$ of a node x is a value that recursively depends on the scores of its neighbors and the probabilities of the connecting edges:

$$\rho(x) \leftarrow \bigotimes_e p_e \cdot \rho(u_e) \tag{1}$$

where e ranges over all incoming edges (u_e, x) . By definition, $\rho(s) = 1$. In Fig. 1a, the propagation score of the target node t is $\rho(t) = p_4\rho(b) \otimes p_5\rho(c) = p_1p_2p_4 \otimes p_1p_3p_5 = 1 - (1 - p_1p_2p_4)(1 - p_1p_3p_5)$. Notice that the propagation score of a node is a recursive or “local property” since it can be calculated from the scores of its neighbors.

With “propagation”, we refer to a family of techniques for calculating the relative importance of nodes in networks with iterative models of computation: “relevance” is propagated across edges from node to node while ignoring past dependencies (see Fig. 2). Thus, unlike reliability, propagation scores can always be computed efficiently, even on very large graphs. Variants of propagation have been successfully used in a range of applications for calculating relevance where exact probabilities are not necessary. Examples include similarity ranking of proteins [77], integrating and ranking uncertain scientific data [16], models of human comprehension [59], activation in feedforward networks [65], search in associative networks [11], trust propagation [38] and influence propagation [35] in social networks, keyword search in databases [5], the noisy-or gate [56, Sect. 4.3.2], computing

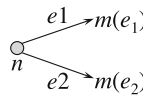
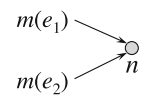
	Messages across edge e	Relevance of node n
		
Pseudo-probabilistic	$m(e_i) \leftarrow p_{e_i} \cdot \rho(n)$ product	$\rho(n) \leftarrow \bigotimes_{e_i} m(e_i)$ independent-or
PageRank	$m(e_i) \leftarrow \frac{1}{d_n} \cdot \rho(n)$ product	$\rho(n) \leftarrow \sum_{e_i} m(e_i)$ addition
Belief propagation	$\mathbf{m}(e_i) \leftarrow \psi_{e_i} \cdot \rho_{\setminus e_i}(n)$ matrix-vector product	$\rho(n) \leftarrow \frac{1}{Z} \bigodot_{e_i} \mathbf{m}(e_i)$ component-wise prod.
Linearized belief prop.	$\mathbf{m}(e_i) \leftarrow \psi_{e_i} \cdot \rho_{\setminus e_i}(n)$ matrix-vector product	$\rho(n) \leftarrow \sum_{e_i} \mathbf{m}(e_i)$ addition

Fig. 2 “Relevance propagation” in graphs works by iteratively calculating messages $m(e)$ across edges e and relevance scores $\rho(n)$ of nodes n . The propagation method we consider is *pseudoprobabilistic* in that the two operators are “independent-and” or product (\cdot), and “independent-or” (\otimes). PageRank and related methods from semi-supervised learning replace the probability p_e of an edge with a weight (here d_n stands for the out-degree of node n) and the independent-or with addition or sum (\sum). Belief Propagation propagates not just one message across an edge but a vector $\mathbf{m}(e)$ of messages, scales this message vector with a matrix ψ_e (also called “edge potential”), and replaces the independent-or with a component-wise product (\odot), followed by a normalization (here Z stands for a normalizer). Linearized Belief Propagation uses again addition as second operator and requires no normalization. Intuitively, the method developed in this paper generalizes pseudoprobabilistic relevance propagation to hypergraphs

web page reputation with PageRank [6], belief propagation in graphical models [56], linearized belief propagation for node labeling [26], or finding true facts from a large amount of conflicting information [78].¹ Note that the resulting relevance scores commonly do not have an exact probabilistic semantics, and may be used as a heuristics instead. For example, the PageRank of a web page does not have to be smaller than 1 (see Fig. 2 for a comparison of the update equations). However, these variants have in common that the score of a node is recursively defined only *in terms of the scores of its neighbors*, and not in terms of the entire topology of the graph.

While Example 1 shows how the propagation score can be defined on graphs, queries are not represented by graphs but *hypergraphs*, in general. 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. Also, the propagation score between two nodes depends on the directionality of the graph, which can be best illustrated with our

¹ Also see [55] for a related discussion of fact finding algorithms, in which the approach of [78] and its use of the iterative propagation Eq. 1 is referred to as “*pseudoprobabilistic*”.

example of k -partite graphs: In Fig. 1a the propagation score from s to t is different from the one from t to s (in fact, the latter coincides with the reliability score). It is not immediately clear what this directionality corresponds to for a relational query whose lineage defines a hypergraph.

With this paper, we introduce a propagation score for queries over PDBs, describe the connection to the reliability score, and give a method to efficiently compute the propagation score for *any self-join-free conjunctive query* with a standard relational database engine. While the propagation score differs from the reliability score, we prove several properties showing that it is a reasonable substitute: (1) propagation and reliability are guaranteed to coincide for all known PTIME queries: our score are thus *strict generalization* of efficient evaluation methods from PTIME to #P-hard queries; (2) propagation is in PTIME and can be evaluated with a *standard relational DBMS* without any changes to the underlying relational query engine; (3) propagation is inspired by the above listed number of successful ranking schemes on graphs: yet our score extends the underlying idea of propagation on graphs to *propagation on hypergraphs*; (4) the propagation score is always an upper bound to the reliability score: it can thus be applied as efficient filter; and (5) the ranking given by the propagation score is very close to the ranking given by the reliability score in our experimental validation.

Example 2 (Example 1 cont.) We have seen that the propagation score differs from the reliability score on the DAG (Directed Acyclic Graph) in Fig. 1a. By inspecting the expressions of the two scores, one can see that they differ in the way 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, dissociation can be obtained by adding a new attribute B to the first relation R (Fig. 1d). The dissociated query is $q^{\Delta_1} :- R^y(s, x, y), S(x, y), T(y, t)$, where the exponent y in R^y indicates the new attribute, and its probability is indeed the same as the propagation score for the graph in Fig. 1a. The important observation here is that, while the evaluation problem for q is #P-hard in general, the query q^{Δ_1} is “hierarchical” [12] and can therefore be computed efficiently. A query q usually has more than one dissociation: q has a second dissociation $q^{\Delta_2} :- R(s, x), S(x, y), T^x(x, y, t)$ 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 $q^{\Delta_3} :- R^y(s, x, y), S(x, y), T^x(x, y, t)$ is a third dissociation. We prove that each dissociation step can only increase the probability (e.g., $r(q) \leq r(q^{\Delta_1}) \leq r(q^{\Delta_3})$). We define the propagation score of q as the smallest probabil-

ity of these three dissociations. The database system has to compute $r(q^{\Delta_1})$ and $r(q^{\Delta_2})$ and return the smallest score: on the graph in Fig. 1a, this is $r(q^{\Delta_2})$, since $r(q) = r(q^{\Delta_2})$.

Contributions and outline. (1) We derive “*query dissociation*” as a generalization of relevance propagation from graphs to hypergraphs and define the propagation score for any self-join-free conjunctive query in terms of dissociations (Sect. 3). A query 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 “hierarchical”*. This is justified by the fact that, in a k -partite graph, the propagation score is precisely the probability of one dissociated hierarchical query. Thus, in our definition, choosing a direction for the network in order to define the propagation score corresponds to choosing a particular dissociation that makes the query hierarchical.

(2) We show how the propagation score can be evaluated with the help of a query-dependent number of query plans (Sect. 4). We achieve this by establishing a one-to-one correspondence between hierarchical dissociations and traditional query plans and showing that *every query plan computes a probability that is an upper bound of query reliability*. Moreover, we describe a natural partial order on the probabilities of query plans. Thus *every self-join-free conjunctive query can be approximated by a fixed number of query plans*, and it suffices to iterate over all minimal plans, compute their probabilities, then take the minimum. We give an intuitive system R-style algorithm [66] that enumerates all minimal plans for a given query q .

(3) We generalize the algorithm to take into consideration schema knowledge on deterministic relations and functional dependencies (Sect. 5). In particular, we give a *unified treatment and generalization of all previously known PTIME self-join free conjunctive queries*, i.e. those that can be evaluated with a query plan in polynomial time in the size of the database, and show that our approach naturally generalizes all known PTIME queries: for every query that is PTIME (whether due to key constraints, or the presence of deterministic tables), reliability and propagation scores always coincide; for every query that is #P-hard, our approach still returns a unique, well-defined score in polynomial time.

(4) We give a set of targeted *multi-query optimization techniques* that considerably speed up the time needed to evaluate the propagation score (Sect. 6). Evaluating some queries may require a large number of plans (e.g., an 8-chain query requires 429 plans). Evaluating all plans sequentially

would still be prohibitively expensive. Instead, we tailor three relational query optimization techniques to dissociation: (i) combining all minimal plans into *one single query*, (ii) reusing *common subexpressions* with views, and (iii) performing *deterministic semi-join reductions*.

(5) We conduct a set of very extensive experiments in which we compare the quality of ranking and scalability of various alternative methods (probabilistic and not) against exact probabilistic inference on TPC-H data [71]. We devise a setup that measures the additional benefit of probabilistic inference for ranking over alternative methods, showing that our technique has high precision for ranking query answers based on their output probabilities. We also show 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 (with 429 query plans) runs only a factor of <10 slower than on a deterministic database.

Prior publications. In recent work [29], we apply the idea of dissociation to both upper and lower bound the probability of Boolean functions, but discuss the connection to query evaluation only in passing. Parts of Sects. 3 and 4 are based on a workshop paper [27]. The remainder is based upon [30]. We added the connection to propagation on graphs, more detailed experiments, slightly changed the formalisms, and included extensive illustrating examples throughout. Due to space restrictions, some of our proofs had to be included in an online appendix on ArXiv [28].

2 Technical background

2.1 Probabilistic databases and self-join-free conjunctive queries

A *tuple-independent probabilistic database* (TI-PDB) is a database D plus a function $p(t) \in [0, 1]$ associating an independent probability to each tuple $t \in D$. We fix a relational vocabulary $\sigma = (R_1, \dots, R_m)$ and denote with D the database, i.e. the collection of tuples and their probabilities. A *possible world* is then a subset of D generated by independently including each tuple t in the world with probability $p(t)$. We use bold notation (e.g., \mathbf{x}) to denote both sets or tuples. A *self-join-free conjunctive query* (sj-free CQ) is a first-order formula $q(\mathbf{z}) = \exists x_1 \dots \exists x_k. (a_1 \wedge \dots \wedge a_m)$ where each atom a_i represents a relation $R_i(\mathbf{x}_i)$, the variables x_1, \dots, x_k are called *existential variables*, and \mathbf{z} are called the *head variables* (or free variables).²

² W.l.o.g. we assume \mathbf{x}_i to be a tuple of only variables and don't write the constants. Selections can always be directly pushed into the database before executing the query.

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, \dots, R_m occurs exactly once in the query. Unless otherwise stated, a “query” in this paper always denotes a sj-free CQ. As usual, we abbreviate a query by $q(\mathbf{z}) :- a_1, \dots, a_m$, and write $\text{HVar}(q) = \mathbf{z}$, $\text{EVar}(q) = \{x_1, \dots, x_k\}$ and $\text{Var}(q) = \text{HVar}(q) \cup \text{EVar}(q)$ for the set of head variables, existential variables, and all variables of q . If $\text{HVar}(q) = \emptyset$, then q is called a *Boolean* query and $\text{EVar}(q) = \text{Var}(q)$. We also write $\text{Var}(a_i)$ for the variables in atom a_i and $\text{at}(x_j)$ for the set of atoms that contain variable x_j . The *active domain* of a variable x_j is denoted ADom_{x_j} ,³ and the active domain of the entire database is $\text{ADom} = \bigcup_j \text{ADom}_{x_j}$. 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. We will refer to this probability as the “*query reliability*” $r(q)$ [36].

It is known that the data complexity [76] of any query q is either in PTIME or #P-hard [13]. The PTIME queries are also called “*safe queries*” and, for the case of sj-free CQs, are characterized precisely by a syntactic property called *hierarchical queries* [12]. We briefly review these results:

Definition 3 (*Hierarchical query*) A query q is called *hierarchical* iff for any two existential variables $x, y \in \text{EVar}(q)$, one of the following three conditions holds: $\text{at}(x) \subseteq \text{at}(y)$, $\text{at}(x) \supseteq \text{at}(y)$, or $\text{at}(x) \cap \text{at}(y) = \emptyset$.

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 4 (Hierarchy dichotomy [12]) *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 $\text{SepVar}(q)$ for the set of existential variables that appear in every atom (called “*separator variables*”). A *connected component* of q (or short, “*query component*”) is a subset of atoms that are connected via existential variables. A query 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 query 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 relational symbols that contain variables from \mathbf{x} . Any query can become disconnected by removing a set of variables.

³ Defined formally as $\text{ADom}_{x_j} = \bigcup_{i: x_j \in \text{Var}(R_i)} \pi_{x_j}(R_i)$.

Lemma 5 (*Hierarchical queries*) *A query q is “hierarchical” iff either: (1) q has a single atom; (2) q has $k \geq 2$ query components all of which are hierarchical; or (3) q has a separator variable x , and $q - \{x\}$ is hierarchical.*

Every hierarchical query can be computed in PTIME, but non-hierarchical queries are #P-hard, in general.⁴

2.2 Probabilistic query plans

Unless otherwise stated, a “query plan” in this paper always denotes a *probabilistic query plan*.

Definition 6 (*Query plans*) A query plan P is given by the grammar $P ::= R_i(\mathbf{x}) \mid \pi_{\mathbf{x}}^P P \mid \bowtie^P [P_1, \dots, P_k]$ where $R_i(\mathbf{x})$ is a relational atom containing the variables \mathbf{x} , $\pi_{\mathbf{x}}^P$ is the *probabilistic project operator with duplicate elimination* (or short “*projection*”), and $\bowtie^P[\dots]$ is the *probabilistic natural join* (or short “*join*”) in prefix notation, which we allow to be k -ary ($k \geq 2$). We require that joins and projections alternate in a plan and do not distinguish between join orders.

We write $\text{Var}(P)$ for all variables in a plan P and $\text{HVar}(P)$ for its *head variables*, which are recursively defined as follows: (1) if $P = R_i(\mathbf{x})$, then $\text{HVar}(P) = \mathbf{x}$; (2) if $P = \pi_{\mathbf{x}}^P(P')$, then $\text{HVar} = \mathbf{x}$; and (3) if $P = \bowtie^P [P_1, \dots, P_k]$, then $\text{HVar}(P) = \bigcup_{i=1}^k \text{HVar}(P_i)$. The *existential variables* $\text{EVar}(P)$ are then defined as $\text{Var}(P) - \text{HVar}(P)$.

Every plan P represents a query q_P defined by taking all atoms mentioned in P as the body and setting $\text{HVar}(q_P) = \text{HVar}(P)$. A plan is called *Boolean* if $\text{HVar}(P) = \emptyset$. We assume the usual sanity conditions on plans to be satisfied: for a projection $\pi_{\mathbf{x}}^P P$ we assume $\mathbf{x} \subseteq \text{HVar}(P)$, and each variable y is projected away at most once in a plan, i.e. there exists at most one operator $\pi_{\mathbf{x}}^P P$ s.t. $y \in \text{HVar}(P) - \mathbf{x}$. For notational convenience, we also use the “*project-away operator*” $\pi_{\mathbf{y}}^P P$ instead of $\pi_{\mathbf{x}}^P P$, where \mathbf{y} are the variables being projected away, i.e. $\mathbf{x} = \text{HVar}(\pi_{\mathbf{y}}^P P) = \text{HVar}(P) - \mathbf{y}$.

Each subplan P returns an intermediate relation of arity $|\text{HVar}(P)| + 1$. The extra *probability attribute* stores a *score*(t) for each output tuple $t \in P(D)$. Given a probabilistic database D and a plan P , *score*(t) is defined inductively on the structure of P as follows: (1) If $t \in R_i(\mathbf{x})$, then *score*(t) = $\mathbb{P}[t]$, i.e. its probability in D ; (2) if $t \in \bowtie^P [P_1(D), \dots, P_k(D)]$ where $t = \bowtie^P [t_1, \dots, t_k]$, then *score*(t) = $\prod_{i=1}^k \text{score}(t_i)$; and (3) if $t \in \pi_{\mathbf{x}}^P P(D)$, and $t_1, \dots, t_n \in P(D)$ are all the tuples that project onto t , then *score*(t) = $\bigotimes_{i=1}^n \text{score}(t_i)$, where “ \otimes ” stands for the independent-or. In other words, *score* computes a probability

⁴ Non-hierarchical queries can be in PTIME when considering functional dependencies or deterministic tables [12,53] (see Sect. 5).

by assuming that all tuples joined by \bowtie^P and all duplicates eliminated by π^P are *independent*. Only if these conditions hold, then score is the correct query probability (also called “query reliability” [36]), but in general it is not. Therefore, score is also called an *extensional semantics* [25, 56, 61] and is, in general, not equal to the query probability, which is defined in terms of possible worlds: $\text{score}(P) \neq \mathbb{P}[q_P]$.⁵ For a Boolean plan P , we get one single score, which we denote $\text{score}(P)$.

The requirement that joins and projections alternate is w.l.o.g. because nested joins, such as $\bowtie^P[\bowtie^P[R_1, R_2], R_3]$ or $\bowtie^P[R_1, \bowtie^P[R_2, R_3]]$, can be rewritten into $\bowtie^P[R_1, R_2, R_3]$ while keeping the same score, e.g., $(p_1 p_2) p_3 = p_1 (p_2 p_3)$. For the same reason we do not distinguish between different permutations in the joins, called join orders [49]. We do not focus on query optimization in this paper until Sect. 6.

Definition 7 (Safe plan) A plan P is called “safe” iff, for each join $\bowtie^P[P_1, \dots, P_k]$, the head variables of each subplan P_i contain the same existential variables of plan P : $\text{HVar}(P_i) \cap \text{EVar}(P) = \text{HVar}(P_j) \cap \text{EVar}(P), \forall 1 \leq i, j \leq k$.

The recursive definition of Lemma 5 gives us immediately a safe plan for a hierarchical query. Conversely, every safe plan defines a hierarchical query. We next illustrate this.

Example 8 (Hierarchical queries and safe plans) Consider $q(x) :- R(x, y), S(x, y, z), T(y, z, u)$, depicted in Fig. 3b with its “augmented incidence matrix”.⁶ The incidence matrix $I(q)$ of a Boolean sj-free CQ q with m atoms and k variables is a $m \times k$ -dimensional 01-matrix with $I(i, j) = 1$ iff $x_j \in \text{Var}(a_i)$. We “augment” it in three ways: (1) We replace 1-entries with circles (o) and ignore 0-entries: this merely cosmetic change makes it easier to recognize patterns; (2) We separate columns for $\text{HVar}(q)$ to the left and $\text{EVar}(q)$ to the right: recall that query components and safety are only determined by $\text{EVar}(q)$. For our example we have $\text{HVar}(q) = \{x\}$ and $\text{EVar}(q) = \{y, z, u\}$; (3) If the query is hierarchical, then we emphasize the hierarchy between $\text{EVar}(q)$ with gray background. For our example we have $\text{at}(u) \subseteq \text{at}(z) \subseteq \text{at}(y)$. Figure 3c shows the corresponding safe plan of q where the hierarchy is reflected in the order

⁵ *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 [56].

⁶ *Incidence matrices* allow us to compactly reason about two types of relationships between variables and relations of sf-free CQs simultaneously: (i) in a column: a variable that is shared across relations, and (ii) in a row: relations that are joined by a variable. They thus allow us to reason about both the “query hypergraph” and the “dual query hypergraph” at the same time, which is helpful also for other types of problems involving sf-free CQs (see, e.g. [24]).

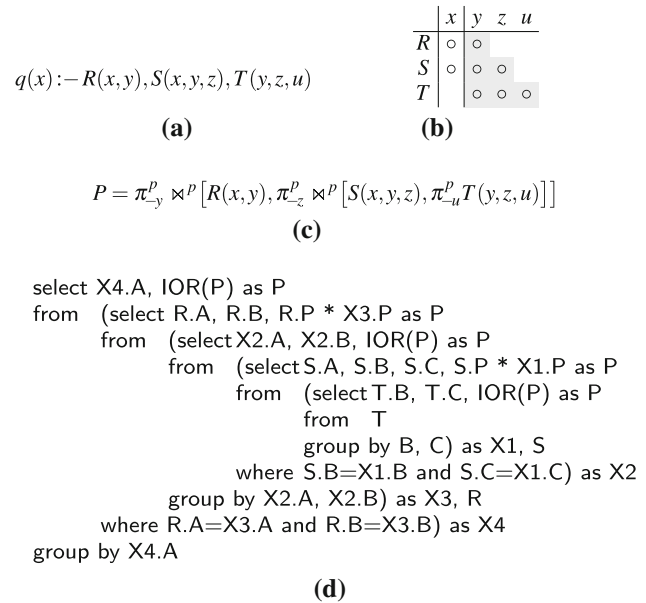


Fig. 3 Example 8. A query q in Datalog (a), its augmented incidence matrix (b), its unique safe plan in our plan notation (c), and in SQL (d)

in which variables are projected away: first u , then z , finally the separator variable y . Figure 3d shows the translation into SQL assuming $R(A, B), S(A, B, C), T(B, C, D)$ as schema and each table having one additional attribute P for the probability of a tuple. Here $\text{IOR}(X)$ is a user-defined aggregate (UDA) that calculates the independent-or for the probabilities of grouped tuples, i.e. $\text{IOR}(p_1, \dots, p_n) = \bigotimes_{i=1}^n p_i$. See [29] for the complete UDA definition in PostgreSQL.

Next consider $q' :- R(x, y), S(x, y, z), T(y, z, u)$, i.e. a variant of q where $x \in \text{EVar}(q')$. Now q' is not hierarchical anymore since $\text{at}(x) \not\subseteq \text{at}(z), \text{at}(x) \not\subseteq \text{at}(y)$, and $\text{at}(x) \cap \text{at}(z) \neq \emptyset$. Starting with P from Fig. 3c and replacing the final projection π_{-y}^P with $\pi_{-x, y}^P$, the plan P' is now unsafe: the join $\bowtie^P [S(x, y, z), \pi_{-u}^P T(y, z, u)]$, has (i) $\text{HVar}(S(x, y, z)) = \{x, y, z\}$, but (ii) $\text{HVar}(\pi_{-u}^P T(y, z, u)) = \{y, z\}$: their intersections with $\text{EVar}(P') = \{x, y, z, u\}$ are now different.

The following proposition summarizes our discussion:

Proposition 9 (Safety [12]) (1) *Let P be a plan for query q . Then $\text{score}(P) = \mathbb{P}[q]$ for any probabilistic database iff P is safe.* (2) *Assuming $\#P \neq \text{PTIME}$, 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 (up to permutation in the join orders), and $\mathbb{P}[q] = \text{score}(P)$.*

2.3 Boolean Formulas

Consider a set of Boolean variables $\mathbf{X} = \{X_1, X_2, \dots\}$ and a probability function $p : \mathbf{X} \rightarrow [0, 1]$. Given a Boolean formula F , denote $\mathbb{P}[F]$ the probability that F is true if each

<table border="1" style="border-collapse: collapse; margin: auto;"> <tr><td style="padding: 2px 5px;">R</td><td style="padding: 2px 5px;">x</td><td style="padding: 2px 5px;">y</td></tr> <tr><td style="padding: 2px 5px;">S</td><td style="padding: 2px 5px;">\circ</td><td style="padding: 2px 5px;">\circ</td></tr> <tr><td style="padding: 2px 5px;">T</td><td style="padding: 2px 5px;">\circ</td><td style="padding: 2px 5px;">\circ</td></tr> </table> <p>(a)</p>	R	x	y	S	\circ	\circ	T	\circ	\circ	<table border="1" style="border-collapse: collapse; margin: auto;"> <tr><td style="padding: 2px 5px;">R</td><td style="padding: 2px 5px;">x</td><td style="padding: 2px 5px;">y</td></tr> <tr><td style="padding: 2px 5px;">S</td><td style="padding: 2px 5px;">\circ</td><td style="padding: 2px 5px;">\bullet</td></tr> <tr><td style="padding: 2px 5px;">T</td><td style="padding: 2px 5px;">\circ</td><td style="padding: 2px 5px;">\circ</td></tr> </table> <p>(b)</p>	R	x	y	S	\circ	\bullet	T	\circ	\circ																											
R	x	y																																												
S	\circ	\circ																																												
T	\circ	\circ																																												
R	x	y																																												
S	\circ	\bullet																																												
T	\circ	\circ																																												
<table style="margin: auto;"> <tr> <td style="padding: 2px 5px;">R</td><td style="padding: 2px 5px;">A</td> <td style="padding: 2px 5px;">S</td><td style="padding: 2px 5px;">A</td><td style="padding: 2px 5px;">B</td> <td style="padding: 2px 5px;">T</td><td style="padding: 2px 5px;">B</td> </tr> <tr> <td style="padding: 2px 5px;">p_1</td><td style="padding: 2px 5px;">a</td> <td style="padding: 2px 5px;">p_2</td><td style="padding: 2px 5px;">a</td><td style="padding: 2px 5px;">b</td> <td style="padding: 2px 5px;">p_4</td><td style="padding: 2px 5px;">b</td> </tr> <tr> <td></td><td></td> <td style="padding: 2px 5px;">p_3</td><td style="padding: 2px 5px;">a</td><td style="padding: 2px 5px;">c</td> <td style="padding: 2px 5px;">p_5</td><td style="padding: 2px 5px;">c</td> </tr> </table> <p>(c)</p>	R	A	S	A	B	T	B	p_1	a	p_2	a	b	p_4	b			p_3	a	c	p_5	c	<table style="margin: auto;"> <tr> <td style="padding: 2px 5px;">R^y</td><td style="padding: 2px 5px;">A</td><td style="padding: 2px 5px;">B</td> <td style="padding: 2px 5px;">S</td><td style="padding: 2px 5px;">A</td><td style="padding: 2px 5px;">B</td> <td style="padding: 2px 5px;">T</td><td style="padding: 2px 5px;">B</td> </tr> <tr> <td style="padding: 2px 5px;">p_1</td><td style="padding: 2px 5px;">a</td><td style="padding: 2px 5px;">b</td> <td style="padding: 2px 5px;">p_2</td><td style="padding: 2px 5px;">a</td><td style="padding: 2px 5px;">b</td> <td style="padding: 2px 5px;">p_4</td><td style="padding: 2px 5px;">b</td> </tr> <tr> <td style="padding: 2px 5px;">p_1</td><td style="padding: 2px 5px;">a</td><td style="padding: 2px 5px;">c</td> <td style="padding: 2px 5px;">p_3</td><td style="padding: 2px 5px;">a</td><td style="padding: 2px 5px;">c</td> <td style="padding: 2px 5px;">p_5</td><td style="padding: 2px 5px;">c</td> </tr> </table> <p>(d)</p>	R^y	A	B	S	A	B	T	B	p_1	a	b	p_2	a	b	p_4	b	p_1	a	c	p_3	a	c	p_5	c
R	A	S	A	B	T	B																																								
p_1	a	p_2	a	b	p_4	b																																								
		p_3	a	c	p_5	c																																								
R^y	A	B	S	A	B	T	B																																							
p_1	a	b	p_2	a	b	p_4	b																																							
p_1	a	c	p_3	a	c	p_5	c																																							

Fig. 4 Example 14: Incidence matrices of $q := R(x), S(x, y), T(y)$ and dissociation $q^\Delta := R^y(x, y), S(x, y), T(y)$. Original database D and new database D^Δ with table R dissociated on variable y . **a** q , **b** q^Δ , **c** D and **d** D^Δ

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 query $q := a_1, \dots, a_m$ on D as the Boolean DNF formula $F_{q,D} = \bigvee_{\theta: \theta \models q} \theta(a_1) \wedge \dots \wedge \theta(a_m)$, where θ ranges over all assignments of $\text{EVar}(q)$ to constants in the active domain that satisfy q on D . It is well known that $\mathbb{P}[q] = \mathbb{P}[F_{q,D}]$. In other words the probability of a Boolean query is the same as the probability of its lineage formula.

Example 10 (Lineage) If $F = XY_1Z_1 \vee XY_2Z_2$, then $\mathbb{P}[F] = p(X)(p(Y_1)p(Z_1) \oplus p(Y_2)p(Z_2))$. Next consider a query $q := R(x), S(x, y), T(y)$ over the database D from Fig. 4c. Then the lineage formula is $F_{q,D} = (R(a) \wedge S(a, b) \wedge T(b)) \vee (R(a) \wedge S(a, c) \wedge T(c))$, i.e. the same as F up to variable renaming. It is now easy to see that $\mathbb{P}[q] = \mathbb{P}[F_{q,D}]$.

A key technique that we use in this paper is the following result from [29]: Let F and F' be two Boolean functions with sets of variables \mathbf{X} and \mathbf{X}' , respectively. We say that F' is a “dissociation” of F if there exists a substitution $\theta : \mathbf{X}' \rightarrow \mathbf{X}$ such that $F'[\theta] = F$. If $\theta^{-1}(X) = \{X', X'', \dots\}$ then we say that the variable X dissociates into X', X'', \dots ; 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 $p : \mathbf{X} \rightarrow [0, 1]$, we extend it to a probability function $p' : \mathbf{X}' \rightarrow [0, 1]$ by setting $p'(X') = p(\theta(X'))$. Then, we have previously shown:

Theorem 11 (Oblivious DNF bounds [29]) *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) $\mathbb{P}[F] \leq \mathbb{P}[F']$, and (2) if every dissociated variable $X \in \mathbf{X}$ is deterministic (i.e. $p(X) = 0$ or $p(X) = 1$), then $\mathbb{P}[F] = \mathbb{P}[F']$.*

Intuitively, a dissociation F' is obtained from a formula F by replacing different occurrences of a variable X with fresh

variables X', X'', \dots ; by doing this, $\mathbb{P}[F']$ gives us an upper bound for $\mathbb{P}[F]$ and may be easier to compute.

Example 12 (Dissociation) $F' = X'Y \vee X''Z$ is a dissociation of $F = XY \vee XZ$, and its probability is $\mathbb{P}[F'] = p(X)p(Y) \oplus p(X)p(Z)$. Here, only the variable X dissociates into X', X'' . It is easy to see that $\mathbb{P}[F] \leq \mathbb{P}[F']$. 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 $F = X$ as $X = XX$. However, $\mathbb{P}[F] = p(X)$, $\mathbb{P}[F'] = p(X)^2$, and thus $\mathbb{P}[F] \not\leq \mathbb{P}[F']$.

3 Dissociation and propagation for unsafe queries

This section defines our technique of “query dissociation” and defines the “propagation score” of a query. Our motivation comes from Theorem 4: hierarchical queries are safe (i.e. in PTIME), while non-hierarchical queries are unsafe (i.e. #P-hard). At its very core, our approach will approximate the probability of a non-hierarchical query with a set of related hierarchical queries. We first define our approach (Sect. 3.1), then draw the connection to propagation in graphs (Sect. 3.2), and finally derive a partial order between a set of hierarchical queries (Sect. 3.3).

3.1 Query dissociation

Definition 13 (Query dissociation) Given a probabilistic database D and a query $q(\mathbf{z}) := R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$. Let $\Delta = (\mathbf{y}_1, \dots, \mathbf{y}_m)$ be a collection of sets of variables with $\mathbf{y}_i \subseteq \text{EVar}(q) - \mathbf{x}_i$ for every relation R_i . The “query dissociation” defined by Δ has then two components:

1. the “dissociated query”:

$$q^\Delta(\mathbf{z}) := R_1^{\mathbf{y}_1}(\mathbf{x}_1, \mathbf{y}_1), \dots, R_m^{\mathbf{y}_m}(\mathbf{x}_m, \mathbf{y}_m)$$

2. the “dissociated database” D^Δ consisting of the tables over the vocabulary $\sigma^\Delta = (R_1^{\mathbf{y}_1}, \dots, R_m^{\mathbf{y}_m})$ obtained by replacing each table R_i with the k_i -fold Cartesian product $R_i \times \text{ADom}_{\mathbf{y}_{i1}} \times \dots \times \text{ADom}_{\mathbf{y}_{ik_i}}$ where $\mathbf{y}_i = (y_{i1}, \dots, y_{ik_i})$. For each new tuple $t' \in R_i^{\mathbf{y}_i}$, its probability is $p'(t') = p(\pi_{\mathbf{x}_i} t')$, i.e. the probability of t in the database D .

Thus, conceptually, we define the semantics of “query dissociation” as follows: Add some existential variables to some atoms in the query; this results in a dissociated query over a new schema. 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. 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 interpretation than the original table. Notice that this is the semantics of a dissociated query, and not the way we actually evaluate queries (in later sections we describe methods that evaluate the dissociated query without modifying the tables in the database).

Example 14 (Example 10 cont.) We illustrate with the query $q := R(x), S(x, y), T(y)$ and the database shown in Fig. 4c where a variable p_i stands for the independent probability of a tuple with index i . Then $\Delta = (\{y\}, \emptyset, \emptyset)$ defines the following dissociation: $q^\Delta := R^y(x, y), S(x, y), T(y)$. Notice we write here and later R^y instead of $R^{\{y\}}$ to simplify our notation. The active domain $ADom_y$ is $\{b, c\}$, and Fig. 4d shows the new database with table R^y as the original table R dissociated on variable y . Notice that the original tuple $R(a)$ got dissociated into two tuples $R^y(a, b)$ and $R^y(a, c)$ with the same probability p_1 . Figure 4b shows q^Δ with the help of an incidence matrix that is *augmented* in a 4th way: while an empty circle (\circ) still indicates that the original relation contains a variable, a full circle (\bullet) now indicates that a relation is dissociated on a variable. Notice that the lineage of the dissociated query q^Δ is $F_{q^\Delta, D^\Delta} = R^y(a, b), S(a, b), T(b) \vee R^y(a, c), S(a, c), T(c)$ and is the same (up to variable renaming) as the dissociation of the lineage of query q : $F' = X'Y_1Z_1 \vee X''Y_2Z_2$. Also notice the deliberate similarity with Example 1 and Fig. 1 from the introduction.

This example generalizes and allows us to prove our first major technical result that query dissociation can only increase the probability:

Theorem 15 (Upper query bounds) *For every database D and every dissociation Δ of a query $q : \mathbb{P}[q^\Delta] \geq \mathbb{P}[q]$.*

Proof (Theorem 15) This follows immediately from Theorem 11 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 $t' \in R_i^{y_i}, \theta(t') = \pi_{x_i}(t')$. \square

By Theorem 4, the probability of a dissociation 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:

Networks (Graphs)	Conjunctive queries (Hypergraphs)
Network reliability: Probability that two nodes are connected. <i>Independent</i> of edge direction.	Query reliability: Probability that query is true in a random world. <i>Independent</i> of query plan.
(+) undirected	(+) undirected
(-) #P-hard	(-) #P-hard
Propagation score: ‘Relatedness’ propagates from source to target. <i>Dependent</i> on edge direction. <i>Upper bound</i> to reliability.	Dissociation score: A query plan evaluates from leafs to root. <i>Dependent</i> on choice of dissociation. <i>Upper bound</i> to reliability.
(-) directed	(-) directed
(+) PTIME	(+) PTIME
	Propagation score: <i>Minimum</i> over all hierarchical dissociations. <i>Unique</i> for given query.
	(+) undirected
	(+) PTIME

Fig. 5 Connection between *reliability* and *propagation* in networks and conjunctive queries (CQs). In contrast to networks, the propagation score for CQs is the minimum over all possible hierarchical dissociations, and is therefore unique for every query and database. (+) and (−) denote positive or negative properties (color figure online)

Definition 16 (Hierarchical dissociation) A dissociation Δ of a query q is called “*hierarchical*” if the dissociated query q^Δ is hierarchical.

The idea now is simple: Find a hierarchical dissociation Δ , compute $\mathbb{P}[q^\Delta]$, and thereby obtain an upper bound on $\mathbb{P}[q]$. In fact, we will consider *all* hierarchical dissociations and take the minimum of their probabilities, since this gives an even better upper bound on $\mathbb{P}[q]$ than that by a single dissociation. We call this quantity the “*propagation score*” of the query q because of similarities with efficient relevance propagation algorithms on graphs. Figure 5 and the next subsection explain in more detail how query dissociation generalizes “*relevance propagation*” from graphs to hypergraphs.

Definition 17 (Propagation) The “*propagation score*” $\rho(q)$ for a query q is the minimum probability of all *hierarchical dissociations*, i.e. $\rho(q) = \min_{\Delta} \mathbb{P}[q^\Delta]$ with Δ ranging over all hierarchical dissociations.

We propose to adopt the *propagation score as an alternative semantics for ranking query results over probabilistic databases*. While the data complexity of 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. Furthermore, $\rho(q) \geq r(q)$ and, if q is safe, then $\rho(q) = r(q)$. Both claims follow immediately from Theorem 15. Hence, the propagation score is a *natural generalization of reliability from safe queries to all queries*: If the query is safe, both scores coincide; if the query is unsafe, propagation still allows to evaluate the query in PTIME (in addition, the next

two sections will show how to evaluate the propagation very efficiently without first dissociating the tables).

3.2 Dissociation and the relation to propagation on graphs

Recall that our original motivation was to develop for queries a concept that is analogous to propagation on directed networks. Queries have no concept of direction, and we suggest that the *choice of direction* in a graph corresponds to a particular *choice of hierarchical dissociation* of a query. We now justify our definitions of query dissociation and propagation by drawing the connection to network reliability and propagation: When a digraph is $k+1$ -partite, then its two terminal reliability can be expressed by a conjunctive k -chain query.⁷ Further, the propagation score over this network corresponds to one of several possible dissociations of this query q , some of which have no natural correspondence to propagation on graphs. Thus, *query dissociation is a strict generalization of network propagation on k -partite graphs*, and we define query propagation as the minimum reliability of a set of query dissociations (see Fig. 5). Notice, however, that dissociation admits a natural interpretation as network propagation only on k -partite graphs, and says nothing about graphs that are not k -partite.

In the following, we use $[k]$ to denote the set $\{1, \dots, k\}$, and $\mathbf{x}_{[i,j]}$ as short form for $(x_i, x_{i+1}, \dots, x_j)$.

Proposition 18 (Connection to networks) *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 $V = \{s\} \cup V_2 \cup \dots \cup V_k \cup \{t\}$, 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 $i \in [k]$. Then:*

(a) *The (s, t) -network reliability of G is $\mathbb{P}[q]$ with:*

$$q := R_1(s, x_2), R_2(x_2, x_3), \dots, R_k(x_k, t)$$

(b) *The directed propagation score from s to t (as defined in Example 1) is $\mathbb{P}[q^\Delta]$ with:*

$$q^\Delta := R_1^{\mathbf{x}_{[3,k]}}(s, \mathbf{x}_{[2,k]}), R_2^{\mathbf{x}_{[4,k]}}(\mathbf{x}_{[2,k]}), \dots, R_k^\emptyset(x_k, t)$$

⁷ A *conjunctive k -chain query* is a query q without self-joins in which each relation is binary, all relations are joined together, and there is no single variable common to more than two relations. Furthermore, the first and last variable are head variables and can be replaced by constants: $q(x_1, x_{k+1}) := R_1(x_1, x_2), R_2(x_2, x_3), \dots, R_k(x_k, x_{k+1})$. The fact that relations are binary entails that the query hypergraph is actually a standard graph. Similarly, the fact that a variable is not common to more than two relations also entails the “dual hypergraph” to be a graph as well. The expression *chain query* derives from the observation that both its hypergraph and dual hypergraph resemble a simple chain.

3.3 Partial dissociation order

The difficulty in computing $\rho(q)$ is that the total number of dissociations is large even for relatively small queries: the number corresponds to the cardinality of the power set of variables that can be added to atoms. Thus, if q has k existential variables and m atoms, then q has $2^{|K|}$ possible dissociations with $K = \sum_{i=1}^m (k - |\text{Var}(a_i)|)$ forming a *partial order* in the shape of a *power set lattice*:

Definition 19 (*Partial dissociation order*) We define the partial order on the dissociations of a query as:

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

Whenever $\Delta \leq \Delta'$, then $q^{\Delta'}$, $D^{\Delta'}$ is a dissociation of q^Δ , D^Δ (given by $\Delta'' = \Delta' - \Delta$). Therefore, we obtain immediately: If $\Delta \leq \Delta'$ then $\mathbb{P}[q^\Delta] \leq \mathbb{P}[q^{\Delta'}]$. However, the statement holds in both directions:

Theorem 20 (*Partial dissociation order*) *For every two dissociations Δ and Δ' of a query q , the following holds over every database:*

$$\Delta \leq \Delta' \Leftrightarrow \mathbb{P}[q^\Delta] \leq \mathbb{P}[q^{\Delta'}]$$

Example 21 (*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 Fig. 6a with the help of augmented incidence matrices. Among the 8 dissociations, 5 are hierarchical, and 2 among those 5 are minimal:

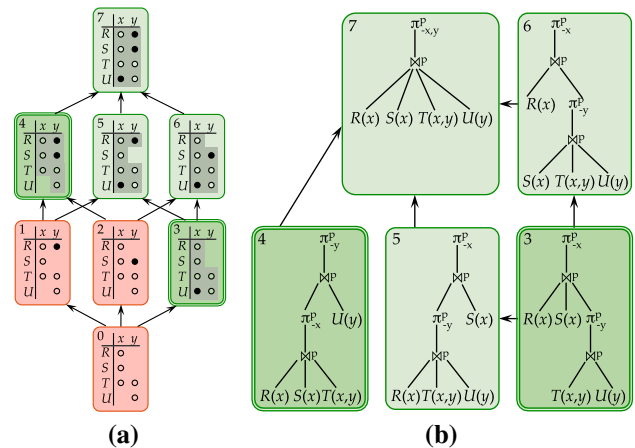


Fig. 6 Examples 21 and 23: **a** Partial dissociation order for $q := R(x), S(x), T(x, y), U(y)$. “Hierarchical dissociations” are green and have the hierarchies between variables shown in their augmented incidence matrices (3–7), “minimal hierarchical dissociations” are dark green and double-lined (3 and 4). **b** All 5 query plans, their correspondence to hierarchical dissociations, and their partial dissociation order (color figure online)

$$q^{\Delta_3} := R(x), S(x), T(x, y), U^x(x, y)$$

$$q^{\Delta_4} := R^y(x, y), S^y(x, y), T(x, y), U(y)$$

The propagation score is the minimum score of all *minimal hierarchical dissociations*: $\rho(q) = \min_{i \in \{3,4\}} \mathbb{P}[q^{\Delta_i}]$. 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 being $\frac{1}{2}$. Then $\mathbb{P}[q] = \frac{83}{29} \approx 0.161$, while $\mathbb{P}[q^{\Delta_3}] = \frac{169}{210} \approx 0.165$ and $\mathbb{P}[q^{\Delta_4}] = \frac{353}{211} \approx 0.172$. Both dissociations give upper bounds, and the propagation score is their minimum (≈ 0.165). Figure 6b is explained later in Example 23.

The smallest element in the lattice of dissociations is $\Delta_{\perp} = (\emptyset, \dots, \emptyset)$ with $q^{\Delta_{\perp}} = q$, and the largest element is $\Delta_{\top} = (\text{Var}(q) - \text{Var}(a_1), \dots, \text{Var}(q) - \text{Var}(a_m)).q^{\Delta_{\top}}$ is always hierarchical as every atom contains all variables. As we move up in the lattice the probability increases, but the hierarchy status may toggle arbitrarily from hierarchical to non-hierarchical and back. For example, the query $q := R(x), S(y), T(x, y, z)$ is non-hierarchical, its dissociation $q' := R(x), S^y(x, y), T(x, y, z)$ is hierarchical, its dissociation $q'' := R^z(x, z), S^y(x, y), T(x, y, z)$ is non-hierarchical again.

This suggests the following naive algorithm for computing $\rho(q)$: Enumerate all dissociations $\Delta_1, \Delta_2, \dots$ by traversing the lattice breadth-first, bottom up (i.e. whenever $\Delta_i < \Delta_j$ 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 $\Delta_j > \Delta_i$. 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 D^{Δ_i} for each hierarchical dissociation Δ_i . In the next two sections we show how to evaluate the propagation score very efficiently.

4 Dissociations and minimal query plans

So far, in order to compute the propagation score of a query q , we need to dissociate its tables and compute several dissociated hierarchical queries q^{Δ} . In practice, we will not apply naively query dissociation (Definition 13) because creating the dissociated database is very inefficient. We next show a 1-to-1 correspondence between hierarchical dissociations and query plans which allows us to calculate $\mathbb{P}[q^{\Delta}]$ on the original database (Sect. 4.1), and then present an efficient algorithm for enumerating a minimum number of query plans we need to evaluate (Sect. 4.2).

4.1 Hierarchical dissociations and query plans

We next show (i) how to efficiently find hierarchical dissociations (by iterating over query plans instead of all dissociations), and (ii) how to compute $\mathbb{P}[q^{\Delta}]$ without having to materialize the dissociated database D^{Δ} .

Theorem 22 (Hierarchical dissociation) *For every sf-free CQ, there is an isomorphism between the set of query plans and the set of hierarchical dissociations. Moreover, the probability of a hierarchical dissociated query q^{Δ} is equal to the score of the corresponding plan: $\mathbb{P}[q^{\Delta}] = \text{score}(P_{\Delta})$.*

We next describe the mappings: (1) $\Delta \mapsto P_{\Delta}$: Consider a hierarchical query dissociation q^{Δ} and denote its corresponding unique safe plan P_{Δ} . This plan uses dissociated relations, hence each relation $R_i^{y_i}(x_i, y_i)$ has extraneous variables y_i . Drop all variables y_i from the relations and all operators using them. Since we only remove existential variables from subgoals, the usual sanity conditions for projections are satisfied and each variable is still projected away in at most one project operator. This transforms P_{Δ} into a regular, generally unsafe plan P for q . For a trivial example, the plan corresponding to the top dissociation Δ_{\top} of a query q (i.e. the dissociation at the top of the partial dissociation order with $y_i = \text{EVar}(q) - x_i$) is $\pi_{\text{EVar}(q)}^P \bowtie^P [P_1, \dots, P_k]$: It first joins all tables, then projects away all existential variables.

(2) $P \mapsto \Delta_P$: Conversely, consider any plan P for q . We define its corresponding hierarchical dissociation Δ_P as follows: For each join operation $\bowtie^P [P_1, \dots, P_k]$, let its *join variables* JVar be the union of the head variables of all subplans: $\text{JVar} = \bigcup_j \text{HVar}(P_j)$. We go from a plan P to a hierarchical dissociation $\Delta = g(P)$ by recursively dissociating each relation R_i occurring in a subplan P_j of a join operation $\bowtie^P [P_1, \dots, P_k]$ on the missing *existential variables* $(\text{JVar} - \text{HVar}(P_j)) \cap \text{EVar}(P)$. Then, recursively and for every relation R_i occurring in P_j , add those variables to y_i .⁸

Example 23 (Example 21 continued) We saw in Example 21 that the query $q := R(x), S(x), T(x, y), U(y)$ has 8 dissociations depicted in Fig. 6a. Among those, 5 are hierarchical, and Fig. 6b shows their query plans:

$$P_{\Delta_3} = \pi_{-x}^P \bowtie^P [R(x), S(x), \pi_{-y}^P \bowtie^P [T(x, y), U(y)]]$$

$$P_{\Delta_4} = \pi_{-y}^P \bowtie^P [U(y), \pi_{-x}^P \bowtie^P [R(x), S(x), T(x, y)]]$$

$$P_{\Delta_5} = \pi_{-x}^P \bowtie^P [S(x), \pi_{-y}^P \bowtie^P [R(x), T(x, y), U(y)]]$$

$$P_{\Delta_6} = \pi_{-x}^P \bowtie^P [R(x), \pi_{-y}^P \bowtie^P [S(x), T(x, y), U(y)]]$$

$$P_{\Delta_7} = \pi_{-x,y}^P \bowtie^P [R(x), S(x), T(x, y), U(y)]$$

⁸ Notice that dissociating a table on any head variable has no implication on the probability of a query result as it does not change its lineage. We therefore only focus on dissociating existential variables.

As every plan corresponds to one hierarchical dissociation, the partial dissociation order carries over to a partial order on all query plans. The propagation score is thus the minimum of the scores of the two minimal plans: $\rho(q) = \min_{i \in \{3,4\}} [score(P_{\Delta_i})]$. Next consider the subplan $\bowtie^P[R(x), T(x, y), U(y)]$ in P_{Δ_3} . Here, $\text{JVar} \cap \text{EVar}(q) = \{x, y\}$ and the corresponding hierarchical dissociation of this subplan is $q^\Delta(x, y) :- R^y(x, y), T(x, y), U^x(x, y)$.

Notice that a hierarchical dissociation is different from and does not imply a safe plan for the original query. It merely states that the dissociated query q^Δ allowed a safe plan P assuming all tuples in its relations were independent. Further notice that while there is a 1-to-1 mapping between hierarchical dissociations and query plans, non-hierarchical dissociations do not correspond to plans and are still hard (e.g., dissociations 0, 1, and 2 in Example 21 and Fig. 6a).

Recall from Sect. 2 that the extensional semantics of an unsafe plan P differs from the query probability: $score(P) \neq \mathbb{P}[q]$, in general. Since we have previously shown that $score(P) = \mathbb{P}[q^\Delta]$ for some dissociation Δ , we derive the following rather surprising result:

Corollary 24 (Query plans are upper bounds) *Let P be any plan for a Boolean query q . Then $\mathbb{P}[q] \leq score(P)$.*

The proof follows immediately from $\mathbb{P}[q] \leq \mathbb{P}[q^{\Delta_P}]$ (Theorem 15) and $\mathbb{P}[q^{\Delta_P}] = score(P)$ (Theorem 22). In other words, any query plan for q as defined in Definition 6 computes a probability score that is guaranteed to be an upper bound on the correct probability $\mathbb{P}[q]$.

4.2 Enumerating minimal query plans

Theorem 22 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 is no need to materialize the dissociated database. However, this approach is still inefficient because it computes several plans that correspond to non-minimal dissociations (e.g., plans 5, 6, 7 in Example 23 are “dominated” by plan 3 since plan 3 is lower in the partial dissociation order). It thus suffices to evaluate only the *minimal query plans*, i.e. those for which the corresponding dissociation is minimal (i.e., not dominated) among all hierarchical dissociations: in our Example 21, these are plans 3 and 4. We next describe our third technical result, the recursive Algorithm 1 that enumerates only the “minimal query plans” (i.e. those that correspond to minimal hierarchical dissociations) and thus the minimum necessary number of query plans to evaluate $\rho(q)$.

We require some additional notation: Call a plan P *minimal* if Δ_P is minimal in the set of hierarchical dissociations.

For example, in Example 21, the minimal plans are P_{Δ_3} and P_{Δ_4} . The propagation score is thus the minimum of the scores of these two plans: $\rho(q) = \min_{i \in \{3,4\}} [score(P_{\Delta_i})]$. 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” is a set of existential variables $\mathbf{x} \in \text{EVar}(q)$ s.t. $q - \mathbf{x}$ is disconnected.⁹ A “*min-cut*” (for *minimal cut*) is a cut for which no strict subset is a cut, i.e. no proper subset $\mathbf{y}' \subset \mathbf{y} \in \text{MinCuts}(q)$ can disconnect the query where $\text{MinCuts}(q)$ denotes the set of all min-cuts. Note that $\text{MinCuts}(q) = \emptyset$ iff q is disconnected.

The connection between $\text{MinCuts}(q)$ 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. $P = \pi_{-\mathbf{x}}^P \bowtie^P [P_1, \dots, P_k]$, and the variables \mathbf{x} projected away are the intersection of the join variables $\text{JVar} = \bigcup_i \text{HVar}(P_i)$ with existential variables, as we must project away all existential variables. We claim that \mathbf{x} is a cut for q and that $q - \mathbf{x}$ has k query components corresponding to P_1, \dots, P_k . Indeed, if P_i, P_j share any common variable y , then they must join on y , hence $y \in \text{JVar}$. Thus, *cuts are in 1-to-1 correspondence with the top-most project-away operator* of a plan. (2) Next suppose that P corresponds to a hierarchical dissociation Δ_P , and let $P' = \pi_{-\mathbf{x}}^{P'} \bowtie^{P'} [P'_1, \dots, P'_k]$ be its unique safe plan. Then $\mathbf{x} = \text{SepVar}(q^{\Delta_P})$; i.e. the top-most project operator removes all separator variables.¹⁰ Furthermore, if $\Delta \succeq \Delta_P$ is a larger hierarchical dissociation, then $\text{SepVar}(q^\Delta) \supseteq \text{SepVar}(q^{\Delta_P})$ (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-cuts*; in other words, $\text{MinCuts}(q)$ is in 1-to-1 correspondence with the top-most projection operator of *minimal plans*.

Our discussion leads immediately to Algorithm 1 for computing the propagation score $\rho(q)$. The algorithm proceeds recursively: If q is a single atom (line 1), then it is safe and we only need to project on the head variables.¹¹ If the query has more than one atom, then we consider two cases depending on whether the query is connected. If the query is discon-

⁹ Recall that we say a query is *connected* if all subgoals are connected by considering only existential variables $\text{EVar}(q)$. In other words, when computing query components we remove head variables from the query: $q - \text{HVar}(q)$. An alternative way to write this is to first substitute all head variables by constants $q' = q[\mathbf{a}/\mathbf{x}]$ (here $q[\mathbf{a}/\mathbf{x}]$ denotes the query obtained by substituting each head variable $x_i \in \mathbf{x}$ with the constant $a_i \in \mathbf{a}$), then to let q_1, \dots, q_k be the components of q' connected by any variable. The query is connected if $k = 1$, otherwise it is disconnected, and $\forall i \neq j : \text{Var}(q_i) \cap \text{Var}(q_j) \subseteq \text{HVar}(q)$.

¹⁰ This follows from the recursive definition of the unique safe plan of a query in Lemma 5: the top-most projection consists precisely of its separator variables.

¹¹ Note that if there are no existential variables ($\mathbf{z} = \mathbf{x}_i$), then there is no need for the projection operator and one could instead simplify to $\mathcal{P} \leftarrow \{R_i(\mathbf{z})\}$, instead of $\mathcal{P} \leftarrow \{\pi_{\mathbf{z}}^P R_i(\mathbf{x}_i)\}$.

```

Recursive algorithm: MP (EnumerateMinimalPlans)
Input: Query  $q(\mathbf{z}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ 
Output: Set of all minimal query plans  $\mathcal{P}$ 
1 if  $m = 1$  then  $\mathcal{P} \leftarrow \{\pi_{\mathbf{z}}^P R_1(\mathbf{x}_1)\};$ 
2 else
3   Set  $\mathcal{P} \leftarrow \emptyset;$ 
4   if  $q$  is disconnected then
5     Let  $q = q_1, \dots, q_k$  be the query components of  $q;$ 
6     foreach  $q_i$  do Let  $\text{HVar}(q_i) \leftarrow \mathbf{z} \cap \text{Var}(q_i);$ 
7     foreach  $(P_1, \dots, P_k) \in \text{MP}(q_1) \times \dots \times \text{MP}(q_k)$  do
8        $\mathcal{P} \leftarrow \mathcal{P} \cup \{\bowtie^P [P_1, \dots, P_k]\};$ 
9   else
10    foreach  $\mathbf{y} \in \text{MinCuts}(q)$  do
11      Let  $q' \leftarrow q$  with  $\text{HVar}(q') \leftarrow \mathbf{z} \cup \mathbf{y};$ 
12      foreach  $P \in \text{MP}(q')$  do  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\pi_{\mathbf{y}}^P P\};$ 

```

Algorithm 1: enumerates all minimal query plans for a query q .

	v	x	y	z	u
R		o	o		o
S			o	o	o
T		o			

(a)

	v	z	y	u	x
R		•	o	o	o
S		o	o	o	o
T		o	o		

(b)

	v	y	u	x	z
R		o	o	o	
S		o	o		o
T		o	•		o

(c)

Fig. 7 Example 26. Query q and its two minimal hierarchical dissociations. Notice the hierarchies between $\text{EVar}(q)$ for both dissociations. **a** q , **b** q^{Δ_1} and **c** q^{Δ_2}

nected (line 4), then the algorithm recursively computes the minimal subplans for each query component, then creates a query plan for each combination of those subplans. If the query is connected (line 8), the algorithm creates a separate plan for each min-cut $\mathbf{y} \in \text{MinCuts}(q)$ by moving \mathbf{y} from the existential variables $\text{EVar}(q)$ to the head variables $\text{HVar}(q)$, thereby disconnecting the query. Notice that recursive calls of the algorithm will alternate between these two cases, until they reach a single atom.

Theorem 25 (Algorithm 1) *Algorithm 1 returns a sound and complete enumeration of minimal query plans.*

Algorithm 1 is sound in that only plans are generated which are not dominated by any other plan. It is complete in that the minimum score of all generated plans is equal to the propagation score of the query.

Example 26 (Enumerate minimal query plans) Consider the non-Boolean query $q(v) :- R(x, y, u), S(y, z, u, v), T(z, v)$. Figure 7a shows its incidence matrix with its head variables $\text{HVar}(q) = \{v\}$ and existential variables $\text{EVar}(q) = \{x, y, z, u\}$ shown separately. The query is connected, and among 2^4 different subsets of $\text{EVar}(q)$, there are 2 “min-cuts”, i.e. minimum sets of variables for which removing them disconnects the query: $\text{MinCuts}(q) = \{\{z\}, \{y, u\}\}$. Projecting away the first min-cut $\{z\}$ separates the query into $q_1(z, v) :- R(x, y, u), S(y, z, u, v)$ and $q_2(z, v) :- T(z, v)$.

Notice that q_1 and q_2 share no existential variables (they only share head variables z and v). Projecting away the second min-cut $\{y, u\}$ separates q into $q_3(y, u) :- R(x, y, u)$ and $q_4(y, u, v) :- S(y, z, u, v), T(z, v)$. Recursive evaluation of q_1 to q_4 shows that they are all hierarchical, from which follows that q has 2 minimal query plans:

$$P_{\Delta_1} = \pi_{-z}^P \bowtie^P [\pi_{-y,u}^P \bowtie^P [\pi_{-x}^P R(x, y, u), S(y, z, u, v)], T(z, v)]$$

$$P_{\Delta_2} = \pi_{-y,u}^P \bowtie^P [\pi_{-x}^P R(x, y, u), \pi_{-z}^P \bowtie^P [S(y, z, u, v), T(z, v)]]$$

Figure 7b and c show their respective hierarchical dissociations, with existential variables re-ordered as to show the hierarchy implied by the query plans.

4.3 Other observations

(1) *Conservation.* 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 Algorithm 1 is *conservative*, in the sense that, if q is safe, then $\rho(q) = \mathbb{P}[q]$. Indeed, in that case $\text{MP}(q)$ returns a single plan, namely the safe P for q , because the empty dissociation, $\Delta_{\perp} = (\emptyset, \dots, \emptyset)$, is safe, and it is the bottom of the dissociation lattice, making it the unique minimal hierarchical dissociation.

(2) *Approximation quality.* We next show that the relative error of approximating query reliability with dissociation improves when the input probabilities decrease. As a consequence, the quality of the ranking also increases. As a practical consequence, the rankings returned by dissociation are better if the input probabilities are small.

Proposition 27 (Small probabilities) *Given a query q and database D . Consider the operation of scaling down the probabilities of all tuples in D with a positive factor $f < 1$. Then the relative error of approximation of the query probability $\mathbb{P}[q]$ by the propagation score $\rho(q)$ decreases as f goes to 0: $\lim_{f \rightarrow 0^+} \frac{\rho(q) - \mathbb{P}[q]}{\mathbb{P}[q]} \rightarrow 0$.*

In the following analytic example, we illustrate Proposition 27 by calculating the relative ratio between propagation and reliability for changing input probabilities.

Example 28 (Small probabilities) We consider the Boolean query $q :- R(x), S^d(x, y), T(y, z)$ and the dissociation $q^{\Delta} :- R^y(x, y), S^d(x, y), T(y, z)$ over the database $r_1 = R(a), s_1 = S^d(a, b), s_2 = S^d(a, c), t_1 = T(b)$, and $t_2 = T(c)$. With deterministic relation S^d , the lineages of q and q^{Δ} are $\text{Lin}(q) = r_1 t_1 \vee r_1 t_2$ and $\text{Lin}(q^{\Delta}) = r_1 t_1 \vee r_1' t_2$, respectively. Assuming $\mathbb{P}[r_1] = r$ and $\mathbb{P}[t_1] = \mathbb{P}[t_2] = t$, the respective probabilities become $\mathbb{P}[q] = r(t \otimes t) = rt(2 - t)$ and $\rho(q) := \mathbb{P}[q^{\Delta}] = rt \otimes rt = rt(2 - rt)$ with r_1 dissociated.

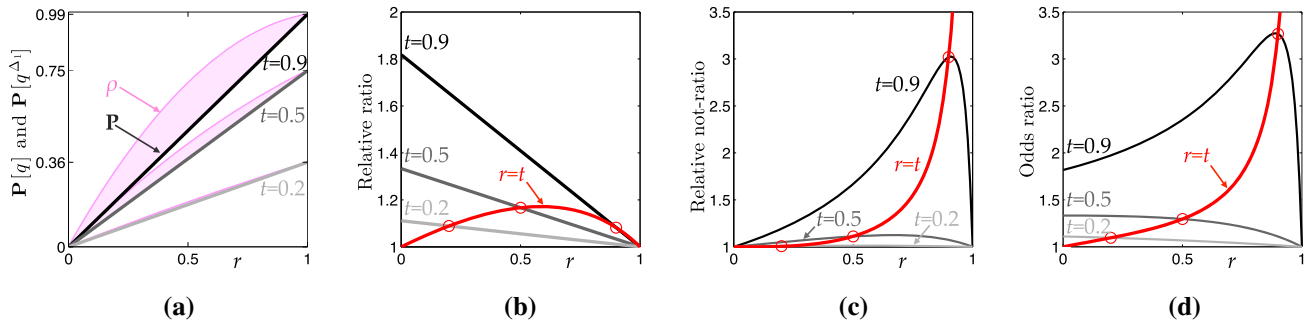


Fig. 8 Example 28. Comparing $\mathbb{P} = \mathbb{P}[q] = \mathbb{P}[r_1 t_1 \vee r_1 t_2]$ and $\rho = \mathbb{P}[q^\Delta] = \mathbb{P}[r_1 t_1 \vee r_1' t_2]$ for varying input probabilities $p = \mathbb{P}[r_1]$ and $q = \mathbb{P}[t_i]$. **a** Probability \mathbb{P} , dissociation ρ . **b** Relative ratio ρ/\mathbb{P} . **c** Relative not-ratio $\bar{\mathbb{P}}/\bar{\rho}$. **d** Odds ratio $(\rho/\bar{\rho})/(\mathbb{P}/\bar{\mathbb{P}})$ (color figure online)

There are four reasonable metrics to measure the approximation quality of dissociation ρ with regard to a probability \mathbb{P} : (1) their *absolute difference* $\rho - \mathbb{P}$, which is not meaningful when both are too close to either 0 or 1; (2) their *relative ratio* ρ/\mathbb{P} , which is not meaningful close to 1; (3) their *relative not-ratio* $\bar{\mathbb{P}}/\bar{\rho}$ with $\bar{x}:=1 - x$, which is not meaningful close to 0; and (4) the *odds ratio* $(\rho/\bar{\rho})/(\mathbb{P}/\bar{\mathbb{P}})$, which is the product of the former two ratios and which is meaningful everywhere in $[0, 1]$. Notice that all four metrics are defined so they are ≥ 1 . Figure 8a shows the original probabilities \mathbb{P} (full lines) and those of their dissociations ρ (border of shaded areas) for various values of r and t . The horizontal axis varies the probability of the dissociated tuple x within $[0, 1]$, and the different lines keep the non-dissociated tuples y_1, y_2 at the same probability either 0.2, 0.5, or 0.9. Figure 8b, c, and d show the approximation quality in terms of our three previously defined ratios. Notice that the red line varies both r and t at the same time by keeping $r = t$. We see that the approximation is good when both input probabilities are small, but get increasingly worse when the probability of the *non-dissociated* variables t_1, t_2 gets close to 1.

Finally notice that the *relative error* $(\rho - \mathbb{P})/\mathbb{P} = \rho/\mathbb{P} - 1 = \frac{t(1-r)}{2-t}$, which clearly tends towards 0 as $r, t \rightarrow 0$.

(3) *Number of minimal query plans.* We end this section by commenting on the number of minimal hierarchical dissociations. Not surprisingly, this number is exponential in the size of the query. To see this, consider a Boolean *k-star query*¹² $q := R_1(x_1), \dots, R_k(x_k), U(x_1, \dots, x_k)$. There are exactly $k!$ minimal hierarchical dissociations: Take any consistent preorder \preceq on the variables. It must be a total preorder, i.e.

¹² A Boolean *conjunctive k-star query* is a query with k unary relations and one k -ary relation: $q := R_1(x_1), \dots, R_k(x_k), U(x_1, \dots, x_k)$. The fact that each variable appears in exactly two relations implies that the dual query hypergraph is actually a standard graph (the dual hypergraph of a query is defined by the relations as vertices and variables as the hyperedges). The expression *star query* derives from the observation that the query's dual (hyper)graph resembles a star with the table U connected to all other relations.

<i>k</i> -star query				<i>k</i> -chain query			
<i>k</i>	#MP	#P	#Δ	<i>k</i>	#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	$2^{(k+1)k}$

Fig. 9 Number of minimal plans, total plans, and total dissociations for star and chain queries (A are OEIS sequence numbers [51])

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 $k!$ such. Note that while the number of hierarchical dissociations is exponential in the size of the query, the number of query plans is independent of the size of the database, and hence our approach has PTIME data complexity [76] for all queries. Figure 9 gives an overview of the number of minimal query plans, total query plans, and dissociations for star and chain queries. Recall that in our definition of query plans, we *do not consider permutations in the joins* (called join orderings [49]). Also, our problem differs from the standard problem of optimal join enumeration in relational database engines. For example, every safe query has only one single minimal query plan, whereas any relational database engine compares several query plans. Later Sect. 6 gives optimizations that allow us to evaluate a large number of plans efficiently.

In summary, our approach allows to rank answers to both safe and unsafe queries in polynomial time in the size of the database, and is conservative w.r.t. the ranking according to exact probabilistic inference for both safe queries and for data-safe queries [42]. The latter follows easily from the point that if a query over a particular database allows one single safe plan, then this plan must be among the minimal plans in the partial dissociation order.

5 Optimizations with schema knowledge

In this section, we show how *deterministic relations* (i.e. all tuples in a relation have probability 1), and *functional dependencies* (e.g., keys) can reduce the number of plans needed to calculate the propagation score.

5.1 Deterministic relations (DRs)

In the following, we denote deterministic relations (DRs) with an exponent “ d ”, i.e. a relation R is probabilistic, and a relation R^d is deterministic. First notice that we can treat DRs just like probabilistic relations, and Corollary 24 with $\mathbb{P}[q] \leq \text{score}(P)$ 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*: $\rho(q) = \min_P \text{score}(P)$. It is known that a non-hierarchical query q can become safe (i.e., $\mathbb{P}[q]$ can be calculated in PTIME with one single plan) if we consider DRs. Thus, 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 29 (Dissociation and DRs) *Dissociating a deterministic relation does not change the probability.*

We thus define a new *probabilistic dissociation preorder* \leq^p that only focuses on probabilistic relations:

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

In other words, $\Delta \leq^p \Delta'$ still implies $\mathbb{P}[q^\Delta] \leq \mathbb{P}[q^{\Delta'}]$, but \leq^p is defined on probabilistic relations only. Notice, that for queries without DRs, the relations \leq^p and \leq coincide. However, for queries with DRs, \leq^p is a preorder, not an order. Therefore, there exist distinct dissociations Δ, Δ' that are equivalent under \leq^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 \leq^p instead of \leq , allows us to further reduce the number of minimal hierarchical dissociations we need to evaluate.

Example 30 (RST query with DRs) Consider the query $q := R(x), S(x, y), T^d(y)$. This query is known to be safe. We thus expect our definition of $\rho(q)$ to find that $\rho(q) = \mathbb{P}[q]$. Ignoring that T^d is deterministic, \leq has two minimal plans corresponding to dissociations $q^{\Delta_1} := R^{\{y\}}(x, y), S(x, y), T^d(y)$, and $q^{\Delta_2} := R(x), S(x, y), T^{d(x)}(x, y)$. Since Δ_2 dissociates only T^d , we now know from Lemma 29 that $\mathbb{P}[q] = \mathbb{P}[q^{\Delta_2}]$. Thus, by using \leq as before, and ignoring information about DRs, we still get the correct answer. However, evaluating the plan P_{Δ_1} is always unnecessary since $\Delta_2 \leq^p \Delta_1$. In contrast, without information about DRs, $\Delta_2 \not\leq^p \Delta_1$, and we would thus have to evaluate both plans.

Figure 10 illustrates these ideas with incidence matrices that are augmented in a 5th way: dissociated variables in

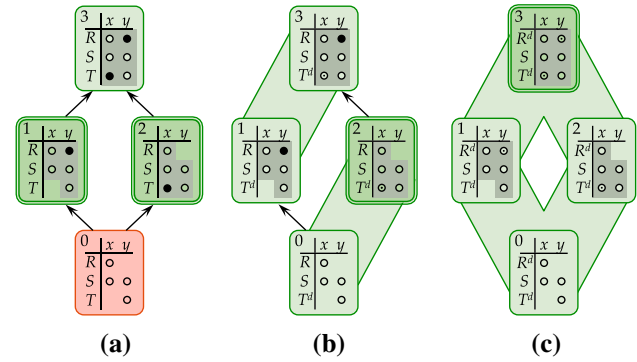


Fig. 10 Example 30. The presence of DRs R^d or T^d changes the *probabilistic dissociation preorder* for $q := R(x), S(x, y), T^d(y)$: several dissociations now have the same probability (shown with shaded areas instead of arrows). Our modified algorithm now returns, for each *minimal safe equivalence class*, the query plan for the top most hierarchical dissociation (shown in dark green and double-lined). **a** No DRs, **b** T^d and **c** R^d and T^d (color figure online)

DRs do not affect the probability and are now marked with dotted circles (\odot) instead of full circles (\bullet). Thus, the preorder \leq^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 all circles. Figure 10b shows that $\rho(q) = \mathbb{P}[q^{\Delta_2}] = \mathbb{P}[q]$ since $\Delta_0 \equiv^p \Delta_2 \leq^p \Delta_1 \equiv^p \Delta_3$ (equivalence under \leq^p is shown with former arrows being replaced by broad connectors). 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: $\Delta_0 \equiv^p \Delta_2$.

Figure 10c shows that for R^d and T^d being deterministic, all three possible query plans (corresponding to Δ_1, Δ_2 , and Δ_3) form an equivalence class in \leq^p with Δ_0 , and thus give the exact probability. In other words, the number of hierarchical dissociations “minimal in \leq^p ” has increased to 3, but all of them are now in the same equivalence class and thus have the same 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 plan for the top hierarchical dissociation in \leq for each minimum safe equivalence class) since P_{Δ_3} least constrains the join order between tables: compare $P_{\Delta_3} = \pi_{x,y}^p \bowtie^p [R(x), S(x, y), T^d(y)]$ with $P_{\Delta_2} = \pi_x^p \bowtie^p [R(x), \pi_y^p \bowtie^p [S(x, y), T^d(y)]]$.

We now explain two simple modifications to Algorithm 1 that achieve our desired optimizations described above:

- (1) Let a “*p-cut*” be a set of existential variables $\mathbf{x} \in \text{EVar}(q)$ s.t. $q - \mathbf{x}$ has at least two query components, each of which has at least one probabilistic table. Denote

by $\text{MinPCuts}(q)$ the set of all “minimal p -cuts” and replace $\text{MinCuts}(q)$ with $\text{MinPCuts}(q)$ in line 10.

- (2) Denote with m_p the number of probabilistic relations in a query, and w.l.o.g. order the relations in a query q as to first list the probabilistic relations, followed by DRs. Replace the stopping condition in line 1 with: **if** $m^p \leq 1$ **then** $\mathcal{P} \leftarrow \{\pi_{\mathbf{x}}^p \bowtie^p [R_1(\mathbf{x}_1), R_2^d(\mathbf{x}_2), \dots, R_m^d(\mathbf{x}_m)]\}$. In other words, if a query has maximal one probabilistic relation, then first join all relations, then project on the head variables.

Theorem 31 (Algorithm 1 with DRs) *Above two modifications to Algorithm 1 return one plan for each minimal safe equivalence class in \preceq^p , i.e. it returns a minimum number of plans to calculate $\rho(q)$ given schema knowledge about deterministic relations.*

Example 32 (Example 30 continued) For our simple query $q := R(x), S(x, y), T^d(y), \text{MinCuts}(q) = \{\{x\}, \{y\}\}$, while $\text{MinPCuts}(q) = \{\{x\}\}$. 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, $\text{MinPCuts}(q) = \emptyset$) and the algorithm returns P_{Δ_3} as single plan (Fig. 10c).

Note that as before, if the query is safe, then the algorithm produces one single query plan. Furthermore, if all relations are deterministic, then the returned query plan consists of one multi-join between all relations followed by a single projection: $\pi_{\mathbf{x}}^p \bowtie^p [R(\mathbf{x}_i), \dots, R(\mathbf{x}_m)]$. The translation into SQL is thus one single standard deterministic SQL query and the query optimizer is unconstrained to determine the optimal join order between the relations. Therefore, *our algorithm conservatively extends deterministic SQL queries to probabilistic SQL queries* in that fully deterministic queries are evaluated exactly as deterministic SQL.

Example 33 (Example 23 continued) First consider relation U to be deterministic: $q_1 := R(x), S(x), T(x, y), U^d(y)$. Figure 11a shows that $\rho(q) = \mathbb{P}[q^{\Delta_3}] = \mathbb{P}[q]$ and thus the query is safe. Put differently, there is only one *minimal safe equivalence class* $\Delta_0 \equiv^p \Delta_3$, and our modified algorithm returns P_{Δ_3} as single minimal plan. Next consider S to be deterministic: $q_2 := R(x), S^d(x), T(x, y), U(y)$. The query is now in an equivalence class with Δ_2 (Fig. 11b). However, neither Δ_0 nor Δ_2 is hierarchical and thus the query is hard. Also, Δ_3 is in a minimal safe equivalence class with Δ_6 , the latter of which has fewer constraints on the joins. Thus, the algorithm returns P_{Δ_4} and P_{Δ_6} as the least constrained plans, one from each minimum safe equivalence class.

We end with a short comment on actual implementation in SQL: In practice, deterministic relations do not have a probabilistic attribute, which simplifies the calculations. Consider a plan $P = \bowtie^p [T^d(z), R^d(x, z), M(x, y, z, u)]$.

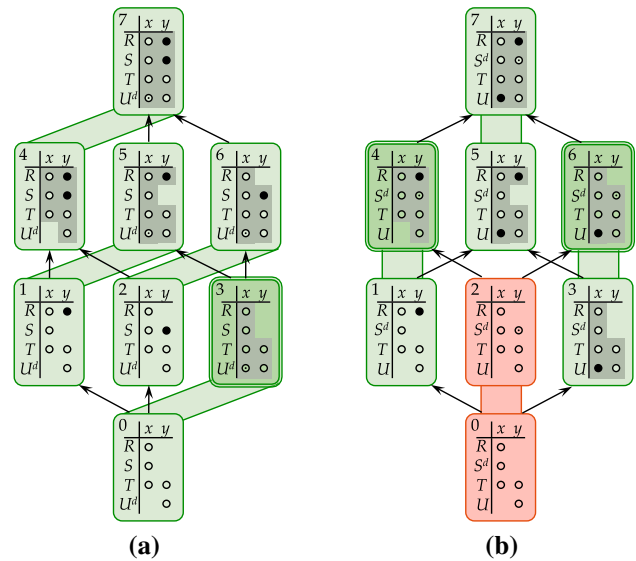


Fig. 11 Example 33 (Example 23 continued). The presence of DRs (either U^d or S^d) changes the probabilistic dissociation preorder and thus the minimal plans returned by our algorithm: P_{Δ_3} , or P_{Δ_4} and P_{Δ_6} . **a** U^d and **b** S^d

This query is specified as join with standard semantics $T(x, y, z, u, p) := T(z), R(x, z), M(x, y, z, u, p)$ over the input relations with p as the probability attribute.

5.2 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 Example 30: 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 $\text{Var}(q)$ 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 under Γ , i.e. \mathbf{x}_i^+ is the smallest set of variables that contains \mathbf{x}_i , and contains z whenever it contains y and $y \leftarrow z$ is a FD in Γ .¹³ Then we show:

Lemma 34 (Dissociation and FDs) *Dissociating an atom $R_i(\mathbf{x}_i)$ on any variable $y \in (\mathbf{x}_i^+ - \mathbf{x}_i)$ does not change the probability of the query.*

In other words, dissociating a table on a variable that is functionally dependent on the existing variables does not change the probability. This lemma is similar to Lemma 29 for DRs. Next, for any query q , denote q^+ the query where each atom

¹³ E.g., if $\mathbf{x} = \{y\}$ and $\Gamma = \{x \rightarrow y, y \rightarrow z, z \rightarrow u\}$, then $\mathbf{x}^+ = \{y, z, u\}$.

$R_i(\mathbf{x}_i)$ is replaced with $R_i(\mathbf{x}_i^+)$, and call q^+ the closure of q . Call a query “closed” if $q^+ = q$, and call a dissociation Δ closed if q^Δ is closed, i.e. $(\mathbf{x}_i \cup \mathbf{y}_i)^+ = (\mathbf{x}_i \cup \mathbf{y}_i)$. We can then further refine our *probabilistic dissociation preorder* \leq^P by:

$$\Delta \leq^P \Delta' \Leftrightarrow \forall i \text{ with } R_i \text{ probabilistic} : (\mathbf{x}_i \cup \mathbf{y}_i)^+ \subseteq (\mathbf{x}_i \cup \mathbf{y}'_i)^+$$

In other words, we only need to consider closed dissociations. As a consequence, using our refined definition of \leq^P allows us to further reduce the number of *minimal safe equivalence classes*. We next state a result by [53] in our notation:

Proposition 35 (Safety and FDs [53, Prop. IV.5]) *A query q is safe under FDs Γ iff q^+ is hierarchical.*

This justifies our third modification to Algorithm 1 for enumerating the minimum number of plans for computing $\rho(q)$ over a database that satisfies FDs Γ and has DRs:

- (3) At each recursive call of Algorithm 1, just before line 1, replace q with its closure q^+ .

Theorem 36 (Algorithm 1 with DRs and FDs) *Above three modifications to Algorithm 1 return one plan for each minimal safe equivalence class in \leq^P , i.e. it returns a minimum number of plans to calculate $\rho(q)$ in the presence of deterministic relations and functional dependencies.*

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* [12,53]. In particular, we can reformulate the known safe query dichotomy [12] in our notation very succinctly:

Corollary 37 (Dichotomy) $\mathbb{P}[q]$ is in PTIME iff there exists a dissociation of q^+ that is hierarchical and that dissociates only deterministic relations. In particular, if all relations are probabilistic then $\mathbb{P}[q]$ is in PTIME iff q^+ is hierarchical.

Corollary 38 (Dichotomy in plans) $\mathbb{P}[q]$ can be calculated in PTIME iff our modified algorithm returns one single plan.

To see what Corollary 37 says, assume first that there are no FDs: Then q is in PTIME iff there exists a hierarchical dissociation Δ that dissociates only DRs. If there are FDs, then we first compute the closure q^+ (called “full chase” in [53]), then apply the same criterion to q^+ .

Example 39 (Example 33 continued) We illustrate here how FDs can change the “probabilistic dissociation preorder”. Analogously to DRs, we mark variables in the incidence matrix that are dissociated as result of an FD and do not affect the probability with a dotted circle (\odot) instead of a bullet (\bullet). As before, the preorder \leq^P is determined entirely

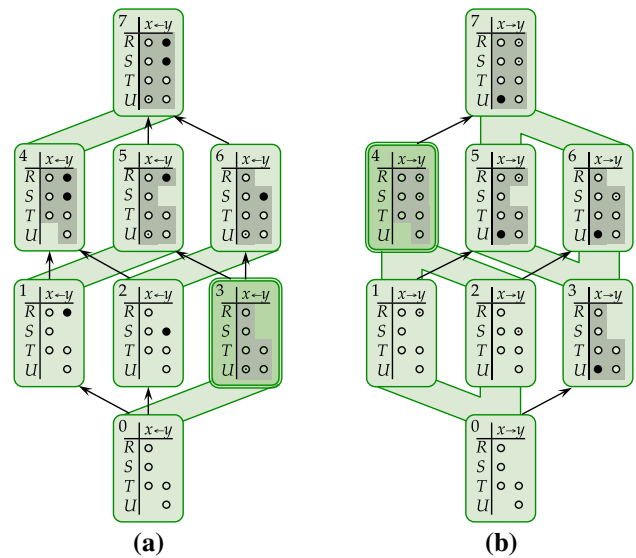


Fig. 12 Example 39: (Example 33 continued): The presence of FDs also changes the probabilistic dissociation preorder and thus the minimal plans returned by our algorithm: either P_{Δ_3} (as in Fig. 11a) or P_{Δ_4} . **a** $\Gamma : y \rightarrow x$. **b** $\Gamma : x \rightarrow y$

by full circles (representing dissociated variables in probabilistic relations that are not implied by FDs on the other variables). However, as before, the correspondence to plans (as implied by the hierarchy between all variables) is still determined by all circles.

First consider $\Gamma : y \rightarrow x$: Fig. 12a shows that this FD leads to the same preorder as for DR U^d from Fig. 11a. Thus, the minimal plan is also P_{Δ_3} . Next consider $\Gamma : x \rightarrow y$: Figure 12b shows that there are now only two equivalence classes, both of which are safe, and one of which is minimal: $\Delta_0 \equiv^P \Delta_1 \equiv^P \Delta_2 \equiv^P \Delta_4$. Among those, only Δ_4 is hierarchical and is thus the one returned by the algorithm.

6 Multi-query Optimizations

So far, we enumerate all minimal query plans, 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. Notice that these three optimizations and the previous optimizations using schema knowledge are orthogonal and can be arbitrarily combined in the obvious way. We use the following example to illustrate the first two optimizations:

Example 40 (Multi-query optimizations) Consider the query $q:- R(x, z), S(y, u), T(z), U(u), M(x, y, z, u)$. Our default is to evaluate all 6 minimal plans returned by Algorithm 1, then take the minimum score (shown in Fig. 13a). Figure 13b

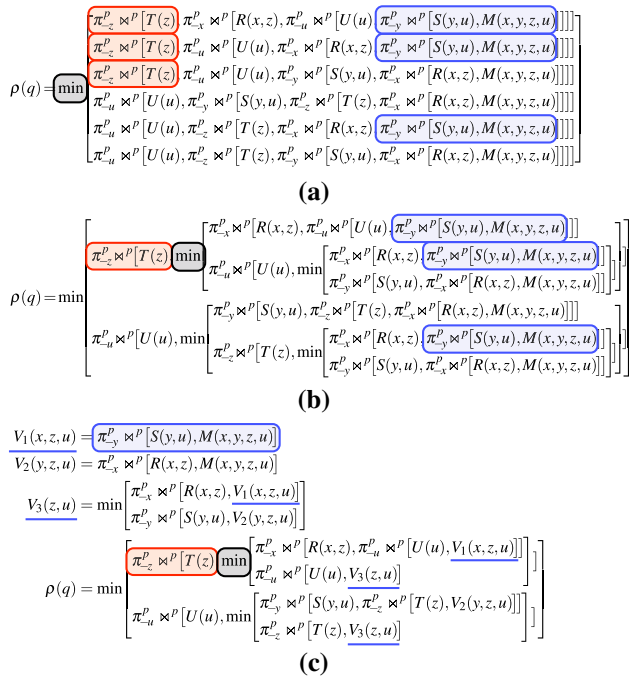


Fig. 13 Example 40 before and after applying optimizations 1 and 2. **a** Result from Algorithm 1: six minimal query plans. **b** Result from Algorithm 2 with Opt. 1: one single query plan. **c** Result from Algorithm 2 plus Opt. 2: re-using common subplans

and c illustrate the optimized evaluations after applying Opt. 1, or Opt. 1 and Opt. 2, respectively.

6.1 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 min-cut $y \in \text{MinCuts}(q)$ in line 12 of Algorithm 1, the adapted Algorithm 2 takes the minimum score over those min-cuts, for each tuple of the head variables in line 10. It thus creates one single query plan. Figure 13b shows this single plan for our running example.

6.2 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 by re-using views. The adapted algorithm works as follows: It first traverses the whole single query plan and remembers each subplan by the atoms used and its head variables in a HashSet. If it sees a subplan twice, it creates a new view for this subplan, map-

```

Recursive algorithm: SP (SinglePlan)
Input: Query  $q(\mathbf{z}) :- R_1(\mathbf{x}_1), \dots, R_{m_p}(\mathbf{x}_{m_p}), \dots, R_m^d(\mathbf{x}_m)$ 
Output: Single query plan  $P$ 
1 if  $m^p \leq 1$  then  $\mathcal{P} \leftarrow \{\pi_x^p \bowtie^p [R_1(\mathbf{x}_1), R_2(\mathbf{x}_2), \dots, R_m^d(\mathbf{x}_m)]\};$ 
2 else
3   if  $q$  is disconnected then
4     Let  $q = q_1, \dots, q_k$  be the query components of  $q$ ;
5     foreach  $q_i$  do  $\text{HVar}(q_i) \leftarrow \text{HVar}(q) \cap \text{Var}(q_i);$ 
6      $P \leftarrow \bowtie^p [\text{SP}(q_1), \dots, \text{SP}(q_k)];$ 
7   else
8     Let  $\text{MinPCuts}(q) = \{y_1, \dots, y_j\};$ 
9     foreach  $y_i$  do  $q'_i \leftarrow q_i$  with  $\text{HVar}(q'_i) \leftarrow \text{HVar}(q) \cup y_i;$ 
10     $P \leftarrow \min [\pi_{y_1}^p \text{SP}(q'_1), \dots, \pi_{y_j}^p \text{SP}(q'_j)];$ 

```

Algorithm 2: Optimization 1 recursively pushes the min operator into the leaves and generates one single query plan.

ping the subplan to a new view definition. The actual plan 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 13c shows the generated views and plans for our running example: Notice that the main plan and the view V_3 both re-use views V_1 and V_2 .

6.3 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 [53], 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.

7 Experiments

We are interested in the efficiency (“how fast?”) and the quality (“how good?”) of ranking by 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 a ranking 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. We use a variant of “Average Precision at 10” defined as $AP@10 := \frac{\sum_{k=1}^{10} P@k}{10}$. Here, $P@k$ is the precision at the k th answer, i.e., the fraction of top k answers according to GT that are also in the top k answers returned. Averaging over several experiments yields MAP [46]. We use a variant of the analytic method proposed in [47] to calculate AP in the presence of ties. As baseline for no ranking, we use “*random average precision*” [16], i.e. we assume all tuples have the same score and are thus tied for the same position.

Exact probabilistic inference. Whenever possible, we calculate GT rankings with a tool called SampleSearch [31, 32], which also serves to evaluate the cost of exact probabilistic inference. We describe the method of evaluating the lineage DNF with SampleSearch in [29].

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 10000 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 clauses in their DNFs. Intuitively, a larger lineage size indicates that an answer has more “support” and should thus be more important. Notice that, in contrast to other methods, we ignore here the weight of support and correlations.

Setup 1. We use the TPC-H DBGEN data generator [71] to generate a 1 GB database to which we add a column P for each table and store it in PostgreSQL 9.2 [58]. 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 intuitive parameterized hard 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

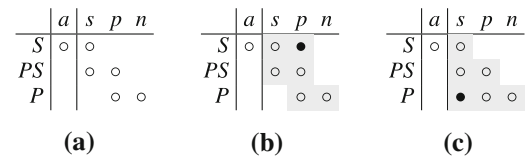


Fig. 14 Parameterized Deterministic SQL query $Q(a)$ over TPC-H. Incidence matrices for TPC-H query $Q(a)$ and its two minimal hierarchical dissociations from either dissociating table S or table P . **a** $Q(a)$, **b** $Q^S(a)$ and **c** $Q^P(a)$

Relations S, PS and P represent tables Supplier, PartSupp and Part, respectively. Variable a stands for attribute nationkey (“answer tuple”), s for supkey, u for partkey (“unit”), and n for name. The probabilistic version of this query is: “Which nations (as determined by the attribute nationkey) are most likely to have suppliers with supkey $\leq \$1$ that supply parts with a 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 $MAP@10 \approx 0.220$. This query has the following two minimal query plans (Fig. 14):

$$P_S(a) = \pi_a^P \bowtie^P [\pi_{a,u}^P \bowtie^P [S(\underline{s}, a), PS(s, u), s \leq \$1], P(\underline{u}, n), n \text{ like } \$2]$$

$$P_P(a) = \pi_a^P \bowtie^P [S(\underline{s}, a), \pi_s^P \bowtie^P [PS(s, u), s \leq \$1, P(\underline{u}, n), n \text{ like } \$2]]$$

Here, P_S and P_P stand for the plans that dissociate tables Supplier or Part, respectively. We take the minimum of the two bounds to determine the propagation score for each answer tuple a . We will also evaluate the speed-up from applying the following deterministic semi-join reduction (Optimization 3) on the input tables and then reusing intermediate query results across both query plans:

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

$$P^*(u, n) :- P(\underline{u}, n), PS^*(s, u)$$

Setup 2. We compare the runtimes 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 return many results, whereas the k -star queries return one tuple, thus representing a Boolean query:

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

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

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 we uniformly draw from the range $\{1, 2, \dots, N\}$. Thus, the parameter N determines the *selectivity* and is varied

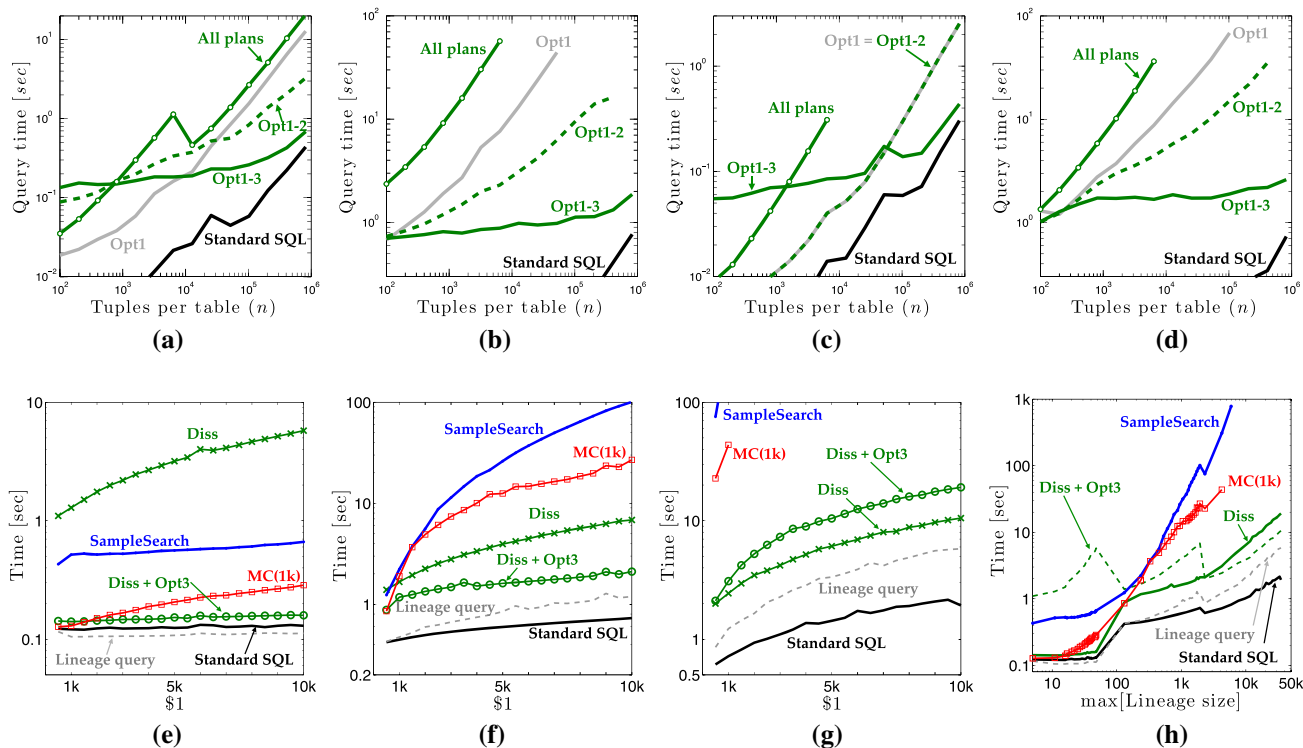


Fig. 15 Timing results: **a–d** For increasing database sizes and constant cardinalities, our optimizations approach deterministic SQL performance. **e–h** For the TPC-H query, the best evaluation for dissociation

is within a factor of 6 of that for deterministic query evaluation. **a** 4-chain query. **b** 7-chain query. **c** 2-star query. **d** 5-star query. **e** \$2 = %red%green%. **f** \$2 = %red%. **g** \$2 = %. **h** Combining **a–c**

as to keep the answer cardinality constant around 20–50 for chain queries, or 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 [48]. To illustrate with numbers, we have to issue 429 query plans in order to evaluate the 8-chain query (see Fig. 9). Each of these plans joins 8 tables in a different order. Optimization 1 then merges those plans together into one truly gigantic single query plan.

7.1 Runtime 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, bringing probabilistic query evaluation close to deterministic evaluation.*

Setup 2. Figure 15a–d show the results for increasing database sizes, and Fig. 16 for increasing query sizes. For example, Fig. 15b shows the performance of computing a 7-chain query which has 132 hierarchical 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 1–3: 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. Without self-join reductions, optimization 1–2 would not execute on the 6-star query with 720 minimal query plans at all (“The query processor ran out of internal resources and could not produce a query plan”). In practice, this suggests that dissociation allows us a large space of optimizations depending on the query and particular database that can conservatively extend the space of optimizations performed today in deterministic query optimizers.

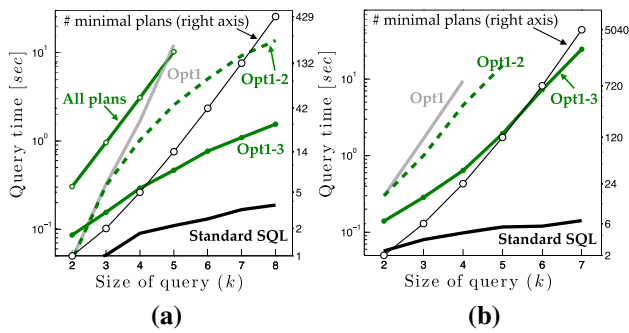


Fig. 16 While the query complexity is exponential (number of minimal plans are shown on the *right side*), our optimizations can even evaluate a very large number of minimal plans (here shown up to 429 for a 8-chain query and 5040 (!) for a 7-star query). **a** k -chain queries. **b** k -star queries

Setup 1. Figure 15e–g compare the running times for 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”). As experimental platform, we use PostgreSQL 9.2 on a 2.5 Ghz Intel Core i5 with 16G of main memory. We run each query 5 times and take the average execution time. We fixed $\$2$ to ‘%red%green%’, ‘%red%’ or ‘%’ and varied $\$1 \in \{500, 1000, \dots, 10k\}$. Figure 15h 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 (Fig. 15e 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 (Fig. 15f 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.*

Figure 15e–h 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 780s for calculating the ground truth for a query with $\max[\text{lin}] = 5.9k$ for which dissociation took 3.0s, and MC(1k) took 42s for a query with $\max[\text{lin}] = 4.2k$ for which dissociation took 2.4s. Dissociation takes only 10.5s for our largest query $\$2 = \text{'%'} and$

	$\$2$	%red%	%green%	%red%	%red%	%	%
	$\$1$	500	10000	500	10000	500	10000
max[lineage size]		5	48	131	1,941	2320	35040
total lineage size		42	1004	2218	44152	40000	800000
SampleSearch [s]		0.43	0.66	1.23	100.71	75.47	—
MC(1k) [s]		0.13	0.29	0.86	26.87	22.75	—
Dissociation & SJ [s]		0.14	0.16	0.88	2.11	2.11	19.14
Dissociation [s]		1.10	5.76	1.39	6.83	2.00	10.52
Lineage SQL [s]		0.12	0.11	0.43	1.19	0.86	5.80
Deterministic SQL [s]		0.12	0.13	0.42	0.73	0.61	1.93

Fig. 17 Overview timing results TPC-H

$\$1 = 10k$ with $\max[\text{lin}] = 35k$. Retrieving the lineage for that query alone takes 5.8s, 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) s to be faster than dissociation (Figure 17).¹⁴

Further optimizations. We found that materialized views performed better than just views. For example, the query $\$1 = 500$ and $\$2 = \text{'%red%green%'}$ takes over 3s with common views instead of our reported 0.88s for materialized views. We also found that using standard database-provided aggregates (which requires us to use the logarithm for products) instead of user-defined aggregates notably speeds up query evaluation for large lineages. Concretely, instead of every occurrence of ‘ior(T.P) as P’ in our queries, we used the following nested PostgreSQL expression: ‘case when (sum(case T.P when 1 then –746 else ln(1 – T.P) end)) < –745 then 1 else 1 – exp(sum(case T.P when 1 then –746 else ln(1 – T.P) end)) end as P’. The outer case statement prevents errors for deterministic tuples (i.e. with $p_i = 1$), and the inner case statement prevents errors due to underflows. As illustration of the improvements, the query $\$1 = 10k$ and $\$2 = \text{'%'}$ would take 42.2s instead of 20.7 with semi-join reduction, and 32.5s instead of 11.3 for the two individual query plans when using a UDF instead of the above expression. We also found that removing the outer case statement would reduce the time by 5% (which could be used if there were no deterministic tuples in a table), and removing the inner case by another 1% (which could be used if there was no risk of underflows). An important by-product of using standard database-defined aggregates is that dissociated queries (and their optimized versions) can be executed with the help of *any standard relational database*, even cloud-based databases that commonly do not allow users to define their own UDAs, e.g. Microsoft SQL Azure. To our best knowledge, this is the currently only technique to approximate rankings of probabilistic queries

¹⁴ 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.

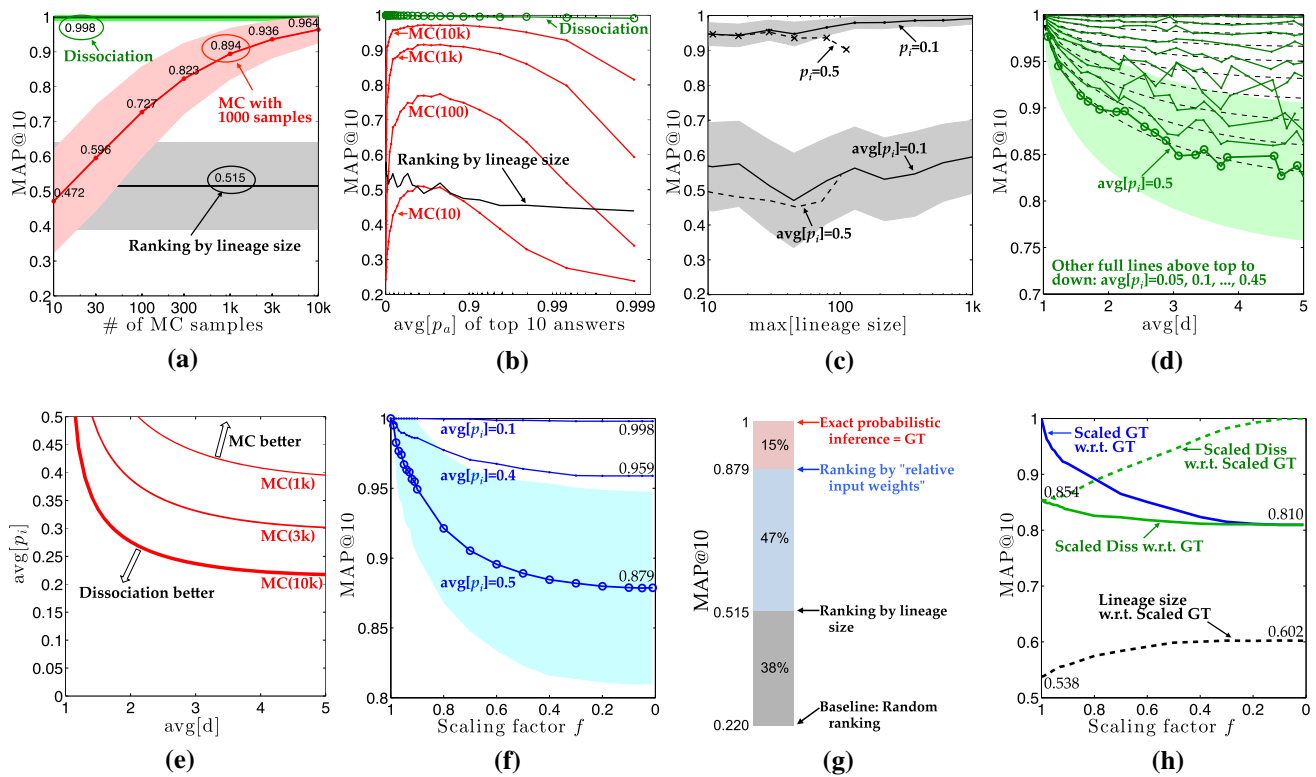


Fig. 18 Ranking experiments on TPC-H: Assumptions for from each plot and conclusions are described below each respective result in the text. **a** Result 3. **b** Result 4. **c** Result 5. **d** Result 6. **e** Result 6. **f** Result 7. **g** Result 7. **h** Result 8 (color figure online)

without any modifications to the database engine nor performing any calculations outside the database.

7.2 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.*

Figure 18a shows averaged results of our probabilistic methods for \$2 = '%r%g%r%a%n%d%'.¹⁵ Shaded areas indicate

¹⁵ Results for MC with other parameters of \$2 are similar. However, the evaluation time for the experiments becomes quickly infeasible.

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 Result 4. Figure 19 gives an overview of the performance of each method, depending on the most important parameters which we will explain next.

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 '%r%e%r%e%'. 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.*

Figure 18b 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

Dissociation	avg[p_i]	avg[d]	MAP@10	stdv
	0.05	5	0.997	0.011
	0.25	2	0.967	0.036
	0.50	1.1	0.968	0.035
	0.50	2	0.894	0.061
	0.50	5	0.833	0.074
MC	avg[p_a]	trials	MAP@10	stdv
	0.1–0.9	10k	0.964	0.040
	0.1–0.9	3k	0.936	0.055
	0.1–0.9	1k	0.894	0.074
	≈ 0.99	10k	0.945	0.046
	≈ 0.99	3k	0.897	0.059
	≈ 0.99	1k	0.827	0.076
Lineage size	p_i		MAP@10	stdv
	random		0.520	0.130
	all equal		0.949	0.033
Random ranking			MAP@10	stdv
			0.220	0.112

Fig. 19 Quality results TPC-H: Our three methods, their respectively most important parameters, and their average ranking qualities

AP over ≈ 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 Fig. 18a.

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

Figure 18c 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 than 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 across all top 10 ranked answer tuples. For all our queries (even those with $\$1 = 10k$ and $\$2 = \%$), $\text{avg}[d]$ 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 $\text{avg}[d]$), 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 $\text{avg}[d]$ in the respective table that gets dissociated. This allows us to draw conclusions for a larger set of parameters. Figure 18d plots MAP values as a function of $\text{avg}[d]$ of the top 10 ranked tuples on the horizontal axis, and various values of $\text{avg}[p_i]$ ($\text{avg}[p_i] = 0.05, 0.10, \dots, 0.5$). Each plotted point averages over at least 10 data points (some have 10, other several thousands). Dashed lines show a fitted parameterized curve to the data points on $\text{avg}[p_i]$ and $\text{avg}[d]$. The figure also shows the standard deviations as shaded areas for $\text{avg}[p_i] = 0.5$. We see that the quality is very dependent on $\text{avg}[p_i]$, as predicted by Proposition 27.

Figure 18e maps the trade-off between dissociation and MC for the two important parameters for the quality of dissociation ($\text{avg}[d]$ and $\text{avg}[p_i]$) 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 Fig. 18a; i.e. assuming $0.1 < \text{avg}[p_a] < 0.9$ for MC. Also recall that for large lineages, having an input probability with $\text{avg}[p_i] = 0.5$ will often lead to answer probabilities close to 1 for which ranking is not possible anymore (recall Fig. 18c). Thus, for large lineages, we need small input probabilities to have meaningful interpretations. And for small input probabilities, dissociation considerably outperforms any other method.

Notice that one should not confuse (i) the AP score of each of the two plans taken separately with (ii) the AP score of the min between the two plans; the ranking produced by the latter can be much better. For example, one experiment ($\$1 = 10k$, and $\$2 = \text{'re\%bl\%re\%'}$) with maximal lineage size 106 has $\text{avg}[d]$ equal 1.053 and 1.099 for P_P and P_S , respectively. None of the two plans gets perfect AP@10. However, using the minimum score of both plans *for each tuple individually* has $\text{avg}[d] = 1.049$ and perfect AP@10 = 1. We also evaluated MAP for ranking all tuples by the plan that has the minimal mean $\text{avg}[d]$ as compared to ranking by the minimum scores for each tuple individually. MAP over all 100k data points would then drop from 0.997 (Fig. 18g) to only 0.995, which shows the value of taking the minimum score for each tuple *individually*.

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.*

This result is a more general statement about the applicability of ranking over probabilistic databases, and motivated by the observation that dissociation works surprisingly well for small input probabilities. Here, we repeatedly evaluated the exact ranking for 7 different parameterized queries over randomly generated databases with one query plan that has $\text{avg}[d] \approx 3$, for two conditions: first on a probabilistic database with $\text{avg}[p_i]$ 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. Figure 18f shows that if all input probabilities are already small (and dissociation already works well), then scaling has little effect on the ranking. However, for $\text{avg}[p_i] = 0.5$ (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 $\text{avg}[p_i] = 0.5$, 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 $f \rightarrow 0$ (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 versus 85% by the lineage size plus the relative weights of input tuples. The remaining 15% come from the actual probabilities (Fig. 18g). While these exact numbers only hold for this particular choice of queries and while the implicit assumption that the quality of ranking were a linear scale of MA is debatable, we think that this “thought experiment” provides an interesting way to think about “the value” of exact probabilistic inference.

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 $\text{avg}[d]$ 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 Fig. 18h, we see that the quality of Scaled Diss w.r.t. Scaled GT $\rightarrow 1$ for $f \rightarrow 0$ since dissociation works increasingly well for small $\text{avg}[p_i]$ (recall Proposition 27). We also see that Scaled Diss w.r.t. GT decreases towards Scaled GT w.r.t. GT for $f \rightarrow 0$. 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.

8 Related Work

Probabilistic databases. Query evaluation over probabilistic databases corresponds to solving the weighted model counting problem, and current approaches can be classified into three categories (Fig. 20): (1) *incomplete approaches* identify tractable cases either at the query-level [12, 13, 23, 53] or the data-level [52, 64, 68] and ignore the rest; (2) *exact approaches* [2, 42, 67] are based on variants and extensions of a complete search based on the DPLL procedure [34] and work well for queries over databases with simple lineage expressions, but perform poorly on complex lineage expressions; and (3) *approximate approaches* usually first compute the lineage of the query on the given database to obtain a Boolean formula, then either apply variants of Monte Carlo sampling methods [41, 44, 45, 62], or approximate the number of models of the Boolean lineage expression [22, 54, 63]. A recent approach combines safe plans with Monte Carlo simulation [37]. An approximate “anytime method” based on DPLL search is developed in [21] that stops the full search whenever a given confidence bound can be guaranteed. This approach allows evaluating a query to a precision determined by a given computational budget. A variant of this method with confidence bounds over first-order lineage formulas is developed in [19]. Our work can be seen as a generalization of some of these techniques: our algorithm returns the exact probability if the query is safe [12, 54] or data-safe [42] and gives a unique and well-defined value for every query. This property can be useful when learning the probabilities from queries. In addition, our method can be used together with any existing relational database without any modifications to the engine. On the other side, our query-centric approach currently works only for self-join-free conjunctive queries and does not allow an iterative refinement or a trade-off

between computational complexity and precision for applications where the exact probability scores are required.

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 [57]. 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 [40], which is the same as a *safe query* in probabilistic databases, and the FO-d-DNNF circuits described in [75] correspond to the safe plans discussed in this paper. See [74] 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* [17]. Quite remarkably, all complex correlations introduced by an MLN can be rewritten into a query over a tuple-independent probabilistic database [33,43,73]. In combination with such rewritings, our techniques can be also applied to MLNs if their rewritings results in conjunctive queries without self-joins.

Dissociation. In a graph-based scenario [16] that basically corresponds to our abstracted Example 1, we observed that propagation-based methods often perform as well as reliability-based methods for predicting protein functions from integrated uncertain biological databases. We then first introduced dissociation in the workshop paper [27] as an attempt to generalize the success of propagation methods from graphs to hypergraphs. [29] provides a general framework for approximating the probability of Boolean functions with both upper and lower bounds. We also illustrate how upper bounds to hard queries can be complemented by lower bounds (those lower bounds, however are not as tight, which is why we only use upper bounds for ranking in this paper). Dissociation is closely related to a number of recent approaches in the graphical model and constraint satisfaction literature which approximate an intractable problem with a tractable relaxed version after *treating multiple occurrences of variables or nodes as independent* or ignoring some equivalence constraints. Those approaches are usually referred to as *relaxation* [72] (see [29] for a detailed discussion on similarities and differences).

9 Conclusions and Outlook

This paper 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 the prop-

	all queries	all data	unique score	performance	rel. algebra	
Safe query plans [12,13]	•	•	•	•	•	} (1) incomplete
Read-once formulas [52,53,68]	•	•	•	•	•	
Exact prob. inference [42]	•	•	•			} (2) slow
Monte Carlo [41,45,62]	•	•	○			
Approx. mod. count. [19,21,54]	•	•	○			} (3) approximate
	•	•	○			

Fig. 20 Current techniques for evaluating probabilistic queries are either (1) *incomplete* and work only on a subset of queries and data instances, or (2) always work but may become arbitrarily *slow* on general data instances, or (3) only *approximate* the actual score

agation score 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. We further showed that the scores for chain queries before and after dissociation correspond to two well-known scoring functions on graphs, namely network reliability (which is #P-hard) and propagation (which is related to PageRank and in PTIME), and that our dissociation scores are thus generalizations of the propagation score from graphs to hypergraphs. We calculated the propagation score by evaluating a fixed number of safe queries, each providing an upper bound on the true probability, then taking their minimum. We provided algorithms that takes into account schema information to enumerate only the minimal necessary plans among all possible plans, and prove our method to be a strict generalization of all known results of PTIME self-join free conjunctive queries. We described 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, IIS-1115188, IIS-1247469, and CAREER IIS-1553547. We like to thank Abhay Jha for help with the experiments in the workshop version of this paper, Alexandra Meliou for suggesting the name “dissociation”, and Vibhav Gogate for guidance in using his tool SampleSearch. WG would also like to thank Manfred Hauswirth for a small comment in 2007 that was crucial for the development of the ideas in this paper.

References

1. Amarilli, A., Amsterdamer, Y., Milo, T.: Uncertainty in crowd data sourcing under structural constraints. In: DASFAA Workshops, pp. 351–359 (2014)
2. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and simple relational processing of uncertain data. In: ICDE, pp. 983–992 (2008)

3. Antova, L., Koch, C., Olteanu, D.: MayBMS: managing incomplete information with probabilistic world-set decompositions. In: ICDE, pp. 1479–1480 (2007)
4. Beame, P., Li, J., Roy, S., Suciu, D.: Model counting of query expressions: limitations of propositional methods. In: ICDT, pp. 177–188 (2014)
5. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using BANKS. In: ICDE, pp. 431–440 (2002)
6. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. *Comput. Netw.* **30**(1–7), 107–117 (1998)
7. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr., E.R., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: AAAI (2010)
8. Chen, Y., Wang, D.Z.: Knowledge expansion over probabilistic knowledge bases. In: SIGMOD, pp. 649–660 (2014)
9. Cohen, W.W.: Data integration using similarity joins and a word-based information representation language. *ACM Trans. Inf. Syst.* **18**(3), 288–321 (2000)
10. Colbourn, C.J.: *The Combinatorics of Network Reliability*. Oxford University Press, New York (1987)
11. Crestani, F.: Application of spreading activation techniques in information retrieval. *Artif. Intell. Rev.* **11**(6), 453–482 (1997)
12. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. *VLDB J.* **16**(4), 523–544 (2007)
13. Dalvi, N.N., Suciu, D.: The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* **59**(6), 30 (2012)
14. Davis, M., Putnam, H.: A computing procedure for quantification theory. *J. ACM* **7**(3), 201–215 (1960)
15. DeepDive: <http://deepdive.stanford.edu/>
16. Detwiler, L., Gatterbauer, W., Louie, B., Suciu, D., Tarczy-Hornoch, P.: Integrating and ranking uncertain scientific data. In: ICDE, pp. 1235–1238 (2009)
17. Domingos, Pedro, Lowd, Daniel: *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool Publishers, San Rafael (2009)
18. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: KDD, pp. 601–610 (2014)
19. Dylla, M., Miliaraki, I., Theobald, M.: Top-k query processing in probabilistic databases with non-materialized views. In: ICDE, pp. 122–133 (2013)
20. Ermis, B., Bouchard, G.: Iterative splits of quadratic bounds for scalable binary tensor factorization. In: UAI, pp. 192–199 (2014)
21. Fink, R., Huang, J., Olteanu, D.: Anytime approximation in probabilistic databases. *VLDB J.* **22**(6), 823–848 (2013)
22. Fink, R., Olteanu, D.: On the optimal approximation of queries using tractable propositional languages. In: ICDT, pp. 174–185 (2011)
23. Fink, R., Olteanu, D.: A dichotomy for non-repeating queries with negation in probabilistic databases. In: PODS, pp. 144–155 (2014)
24. Freire, C., Gatterbauer, W., Immerman, N., Meliou, A.: The complexity of resilience and responsibility for self-join-free conjunctive queries. *PVLDB* **9**(3), 180–191 (2015)
25. Fuhr, N., Rölleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.* **15**(1), 32–66 (1997)
26. Gatterbauer, W., Günnemann, S., Koutra, D., Faloutsos, C.: Linearized and single-pass belief propagation. *PVLDB* **8**(5), 581–592 (2015)
27. Gatterbauer, W., Jha, A.K., Suciu, D.: Dissociation and propagation for efficient query evaluation over probabilistic databases. In: Proceedings of 4th International VLDB workshop on Management of Uncertain Data (MUD), pp. 83–97 (2010)
28. Gatterbauer, W., Suciu, V.: Dissociation and propagation for approximate lifted inference with standard relational database management systems (2013). [arXiv:1310.6257](https://arxiv.org/abs/1310.6257) [cs.DB]
29. Gatterbauer, W., Suciu, D.: Oblivious bounds on the probability of Boolean functions. *ACM Trans. Database Syst. (TODS)* **39**(1), 5 (2014)
30. Gatterbauer, W., Suciu, D.: Approximate lifted inference with probabilistic databases. *PVLDB* **8**(5), 629–640 (2015)
31. Gogate, V., Dechter, R.: SampleSearch: importance sampling in presence of determinism. *Artif. Intell.* **175**(2), 694–729 (2011)
32. Gogate, V., Domingos, P.: Formula-based probabilistic inference. In: UAI, pp. 210–219 (2010)
33. Gogate, V., Domingos, P.: Probabilistic theorem proving. In: UAI, pp. 256–265 (2011)
34. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting. In: *Handbook of Satisfiability*, pp. 633–654 (2009)
35. Goyal, A., Bonchi, F., Lakshmanan, L.V.S.: Learning influence probabilities in social networks. In: WSDM, pp. 241–250 (2010)
36. Grädel, E., Gurevich, Y., Hirsch, C.: The complexity of query reliability. In: PODS, pp. 227–234 (1998)
37. Gribkoff, E., Suciu, D.: Slimshot: in-database probabilistic inference for knowledge bases. *PVLDB* **9**(7), 552–563 (2016)
38. Guha, R.V., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of trust and distrust. In: WWW, pp. 403–412 (2004)
39. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.* **194**, 28–61 (2013)
40. Jaeger, M., Van den Broeck, G.: Liftability of probabilistic inference: upper and lower bounds. In: StaRAI (2012)
41. Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C.M., Haas, P.J.: MCDB: a Monte Carlo approach to managing uncertain data. In: SIGMOD, pp. 687–700 (2008)
42. Jha, A., Olteanu, D., Suciu, D.: Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In: EDBT, pp. 323–334 (2010)
43. Jha, A., Suciu, D.: Probabilistic databases with MarkoViews. *PVLDB* **5**(11), 1160–1171 (2012)
44. Joshi, S., Jermaine, C.M.: Sampling-based estimators for subset-based queries. *VLDB J.* **18**(1), 181–202 (2009)
45. Kennedy, O., Koch, C.: PIP: a database system for great and small expectations. In: ICDE, pp. 157–168 (2010)
46. Manning, C.D., Raghavan, P., Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press, New York (2008)
47. McSherry, F., Najork, M.: Computing information retrieval performance measures efficiently in the presence of tied scores. In: ECIR, pp. 414–421 (2008)
48. Microsoft SQL Server 2012. <http://www.microsoft.com/sqlserver>
49. Moerkotte, G.: Building query compilers. Draft version 03 Mar 2009
50. Niu, F., Ré, C., Doan, A., Shavlik, J.W.: Tuffy: scaling up statistical inference in markov logic networks using an RDBMS. *PVLDB* **4**(6), 373–384 (2011)
51. OEIS: The on-line encyclopedia of integer sequences: <http://oeis.org/>
52. Olteanu, D., Huang, J.: Using OBDDs for efficient query evaluation on probabilistic databases. In: SUM, pp. 326–340 (2008)
53. Olteanu, D., Huang, J., Koch, C.: Sprout: lazy vs. eager query plans for tuple-independent probabilistic databases. In: ICDE, pp. 640–651 (2009)
54. Olteanu, D., Huang, J., Koch, C.: Approximate confidence computation in probabilistic databases. In: ICDE, pp. 145–156 (2010)
55. Pasternack, J., Roth, D.: Knowing what to believe (when you already know something). In: COLING, pp. 877–885 (2010)
56. Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo (1988)

57. Poole, D.: First-order probabilistic inference. In: IJCAI, pp. 985–991 (2003)
58. PostgreSQL 9.2. <http://www.postgresql.org/download/>
59. Quillian, M.R.: Semantic memory. In: Semantic Information Processing, pp. 227–270. MIT Press (1968)
60. Raghunathan, R., De, S., Kambhampati, S.: Bayesian networks for supporting query processing over incomplete autonomous databases. *J. Intell. Inf. Syst.* **42**(3), 595–618 (2014)
61. Ré, C., Dalvi, N.N., Suciu, D.: Query evaluation on probabilistic databases. *IEEE Data Eng. Bull.* **29**(1), 25–31 (2006)
62. Ré, C., Dalvi, N.N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: ICDE, pp. 886–895 (2007)
63. Ré, C., Suciu, D.: Approximate lineage for probabilistic databases. *PVLDB* **1**(1), 797–808 (2008)
64. Roy, S., Perduca, V., Tannen, V.: Faster query answering in probabilistic databases using read-once functions. In: ICDT, pp. 232–243 (2011)
65. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Parallel distributed processing: explorations in the microstructure of cognition, vol. 1, pp. 318–362. MIT Press (1986)
66. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: SIGMOD, pp. 23–34 (1979)
67. Sen, P., Deshpande, A.: Representing and querying correlated tuples in probabilistic databases. In: ICDE, pp. 596–605 (2007)
68. Sen, P., Deshpande, A., Getoor, L.: Read-once functions and query evaluation in probabilistic databases. *PVLDB* **3**(1), 1068–1079 (2010)
69. Singh, A.P., Gordon, G.J.: Relational learning via collective matrix factorization. In: KDD, pp. 650–658 (2008)
70. Stoyanovich, J., Davidson, S.B., Milo, T., Tannen, V.: Deriving probabilistic databases with inference ensembles. In: ICDE, pp. 303–314 (2011)
71. TPC-H Benchmark. <http://www.tpc.org/tpch/>
72. Van den Broeck, G., Choi, A., Darwiche, A.: Lifted relax, compensate and then recover: from approximate to exact lifted probabilistic inference. In: UAI, pp. 131–141 (2012)
73. Van den Broeck, G., Meert, W., Darwiche, A.: Skolemization for weighted first-order model counting. In: KR (2014)
74. Van den Broeck, G., Suciu, D.: Lifted probabilistic inference in relational models. In: UAI tutorials (2014)
75. Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., De Raedt, L.: Lifted probabilistic inference by first-order knowledge compilation. In: IJCAI, pp. 2178–2185 (2011)
76. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: STOC, pp. 137–146 (1982)
77. Weston, J., Elisseeff, A., Zhou, D., Leslie, C.S., Noble, W.S.: Protein ranking: from local to global structure in the protein similarity network. *Proc Natl Acad Sci USA* **101**(17), 6559–6563 (2004)
78. Yin, X., Han, J., Philip, S.Y.: Truth discovery with multiple conflicting information providers on the web. *IEEE Trans. Knowl. Data Eng.* **20**(6), 796–808 (2008)
79. Zeng, K., Gao, S., Mozafari, B., Zaniolo, C.: The analytical bootstrap: a new method for fast error estimation in approximate query processing. In: SIGMOD, pp. 277–288 (2014)
80. Zhang, C., Ré, C.: Towards high-throughput Gibbs sampling at scale: a study across storage managers. In: SIGMOD, pp. 397–408 (2013)