

# Dissociation and Propagation for Efficient Query Evaluation over Probabilistic Databases

Wolfgang Gatterbauer · Dan Suciu

arXiv:1310.6257v2 [cs.DB] 12 Aug 2014

**Abstract** Probabilistic inference over large data sets is an increasingly important data management challenge. The central problem is that exact inference is generally #P-hard, which limits the size of data that can be efficiently queried. This paper proposes a new approach for *approximate evaluation of queries over probabilistic databases*: in this approach, every query is evaluated entirely in the database engine by evaluating a fixed number of query plans, each providing an upper bound on the true probability, then taking their minimum. We provide an algorithm that takes into account important schema information to enumerate only the minimal necessary plans among all possible plans. Importantly, this algorithm is a strict generalization of all known results of PTIME self-join free conjunctive queries: the query is safe if and only if our algorithm returns one single plan. Furthermore, our approach is a generalization of a family of efficient network ranking functions from graphs to hypergraphs. We also describe three relational query optimization techniques that allow us to evaluate all minimal safe plans in a single query and very fast. We give a detailed experimental evaluation of our approach and, in the process, provide 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 statistical relational models since lifted inference corresponds to safe plans in probabilistic databases.

**Keywords** Probabilistic databases · Approximate lifted inference · Ranking semantics · Multi-query optimization

---

Wolfgang Gatterbauer  
Tepper School of Business  
Carnegie Mellon University  
E-mail: [gatt@cmu.edu](mailto:gatt@cmu.edu)

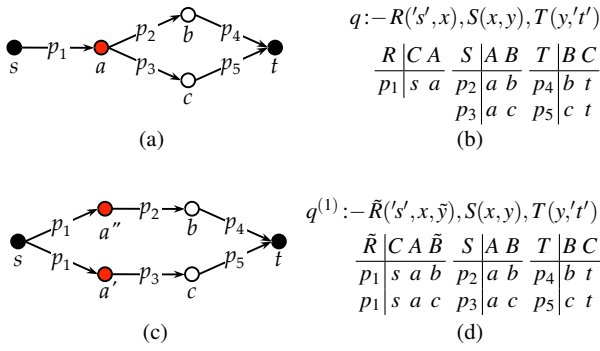
Dan Suciu  
Department of Computer Science & Engineering  
University of Washington

## 1 Introduction

Probabilistic inference over large data sets is becoming a central data management problem. Recent large knowledge bases, such as Nell [10], DeepDive [17], Yago [39], or Google’s Knowledge Vault [23], have millions to billions of uncertain tuples. Data sets with missing values are often “completed” using inference in graphical models [11, 57, 68] or sophisticated low rank matrix factorization techniques [24, 67], which ultimately results in a large, probabilistic database. Data sets that use crowdsourcing are also uncertain [4]. And probabilistic databases have also been applied to bootstrapping over samples of data [73].

However, probabilistic inference is known to be #P-hard in the size of the database, even for some very simple queries [15]. Today’s state of the art inference engines use either sampling-based methods or are based on some variant of the DPLL algorithm for weighted model counting. For example, Tuffy [49], 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 [74]. For another example, MayBMS [6] and its successor Sprout [51] use query plans to guide a DPLL-based algorithm for weighted model counting [34]. While both approaches deploy some advanced relational optimization techniques, at their core they are based on general purpose probabilistic inference techniques, which either run in exponential time (DPLL-based algorithms have been proven recently to take exponential time even for queries computable in polynomial time [7]), or require many iterations until convergence.

In this paper, we propose a different approach to query evaluation on probabilistic databases (PDB). 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, proba-

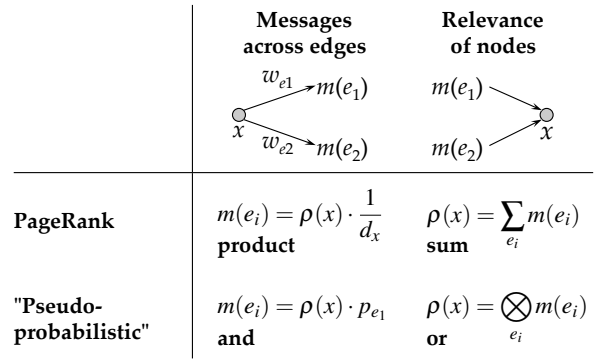


**Fig. 1** Example 1. The *propagation score*  $\rho(t)$  in graph (a) corresponds to the *reliability score*  $r(t)$  in graph (c) with node  $a$  dissociated. (b,d): Corresponding chain queries with respective database instances.

bilistic 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, like cost-based optimizations, multi-core query processing, shared-nothing parallelization are immediately available to queries over probabilistic databases. We achieve this by replacing 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* [12], 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  $t$ :  $r(t) = p_1(1 - (1 - p_2p_4)(1 - p_3p_5))$ . While reliability can be computed efficiently for series-parallel graphs as in Fig. 1a, it is #P-hard in general, even on 4-partite networks [12]. The probability of a query over a PDB corresponds precisely to network reliability. For example, in the case of a 4-partite graph, reliability is given by the probability of the chain query  $q: -R(s, x), S(x, y), T(y, t)$  over the PDB shown in Fig. 1b (we use the terms query reliability and query probability interchangeably in this paper). Notice that the reliability of a node is a *global* property of the entire graph. In contrast, the



**Fig. 2** *Relevance propagation* in graph works by iteratively calculating messages across edges and relevance scores of nodes. The propagation we consider is *pseudoprobabilistic* in that the two operations are a probabilistic-and ( $\cdot$ ) and an independent-or ( $\bigotimes$ ); the latter combines two probabilities as if calculating the disjunction between two independent events and is defined as  $p_1 \otimes p_2 := 1 - (1 - p_1)(1 - p_2)$ .

*propagation score* of a node  $x$  is a value that recursively depends on the scores of its parent nodes and the probabilities of the incoming edges:

$$\rho(x) = 1 - \prod_e (1 - \rho(y_e) \cdot p_e) \quad (1)$$

where  $e$  is the edge  $(y_e, x)$ . By definition,  $\rho(s) = 1$ . In our example, the propagation score of the target node  $t$  is  $\rho(t) = 1 - (1 - p_1p_2p_4)(1 - p_1p_3p_5)$ . Notice that the propagation score of a node is a *local* property since it depends only on its parents. ■

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 [71], integrating and ranking uncertain scientific data [21], models of human comprehension [56], activation in feedforward networks [63], search in associative networks [14], trust propagation [37] and influence propagation [35] in social networks, keyword search in databases [8], the noisy-or gate [54, Sect. 4.3.2], computing web page reputation with PageRank [9], node labeling in graphs [28], or finding true facts from a large amount of conflicting information [72]. Note that the thus calculated relevance scores commonly do not have an exact probabilistic semantics, and may be used as a heuristics instead.<sup>1</sup> For example, the PageRank of a web

<sup>1</sup> For example, the PageRank of a web page is not necessarily smaller than 1. See [53] for a related, recent discussion of fact finding algorithms, in which the approach of [72] and its use of the iterative propagation formula Equation 1 is referred to as “pseudoprobabilistic.”

page is not necessarily smaller than 1. However, they 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 example of  $k$ -partite graphs: In Fig. 1a the propagation score from  $s$  to  $t$  is different from that from  $t$  to  $s$  (in fact, the latter coincides with the reliability score). It is unclear what this directionality corresponds to for arbitrary queries with arbitrary hypergraphs.

With this paper, we introduce a propagation score for queries over PDBs, describe the connection to the reliability score, and give a method to compute the propagation score for *every self-join-free conjunctive query* efficiently 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: (i) propagation and reliability are guaranteed to coincide for all safe queries: our score are thus *strict generalization* of efficient evaluation methods from safe to unsafe queries, (ii) propagation is in PTIME. In addition, we show that every query can be evaluated with a *standard relational DBMS* without any changes to the underlying relational query engine; (iii) 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*; (iv) the propagation score is always an upper bound to the reliability score: it can thus be applied as efficient filter; and (v) the ranking given by the propagation score is very close to the ranking given by the reliability score in our experimental validation.

*Example 1 (continued)* 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 instance, dissociation can be obtained by adding a new attribute  $B$  to the first relation  $R$  (Fig. 1d). The dissociated query is  $q^{(1)}: -\tilde{R}(s, x, \tilde{y}), S(x, y), T(y, t)$ , where the tilde symbol ( $\tilde{\cdot}$ ) indicates dissociated relations or variables with  $\tilde{y} \equiv y$ , 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 because it is an unsafe query [15], the query  $q^{(1)}$  is safe and can therefore be computed efficiently. A query  $q$  usually has more than one dissociation:  $q$  has a second dissociation  $q^{(2)}: -R(s, x), S(x, y), \tilde{T}(\tilde{x}, y, t)$  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  $q^{(3)}: -\tilde{R}(s, x, \tilde{y}), S(x, y), \tilde{T}(\tilde{x}, y, t)$  is a third dissociation. We prove that each dissociation step can only increase the probability, hence  $r(q) \leq r(q^{(1)}) \leq r(q^{(3)})$  and  $r(q) \leq r(q^{(2)}) \leq r(q^{(3)})$ . We define the propagation score of  $q$  as the smallest probability of these three dissociations. The database system has to compute  $r(q^{(1)})$  and  $r(q^{(2)})$  and return the smallest score: on the graph in Fig. 1a, this is  $r(q^{(2)})$ , since  $r(q) = r(q^{(2)})$ . ■

*Main contributions.* Our first main contribution is defining the propagation score for any self-join-free conjunctive query in terms of *dissociations* (Section 3). A dissociation is a rewriting of both the data and the query. On the data, a dissociation is obtained by making multiple, independent copies of some of the tuples in the database. Technically, this is achieved by extending the relational schema with additional attributes. On a query, a dissociation extends atoms with additional variables. We prove that a dissociation can only increase the probability of a query, and define *the propagation score of a query as the minimum reliability of all dissociated queries that are safe*. This is justified by the fact that, in a  $k$ -partite graph, the propagation score is precisely the probability of one dissociated safe query. Thus, in our definition, choosing a direction for the network in order to define the propagation score corresponds to choosing a particular dissociation that makes the query safe. Safe queries [15] can be evaluated efficiently on any probabilistic database, and we show that every query (safe or not) admits at least one safe dissociation. Therefore, the propagation score can always be computed in PTIME.

Our second main contribution is establishing a one-to-one correspondence between safe dissociations and traditional query plans (Section 4). We show that *every query plan computes a probability that is an upper bound of query reliability*. Moreover, we describe a natural partial order on the plans that guarantees that their probabilities satisfy this partial order. Thus, in order to obtain the propagation score of a query, it suffices to iterate over all minimal plans, compute their probabilities, then take the minimum. We give an intuitive system R-style algorithm [64] that enumerates all plans that correspond to minimal safe dissociations.

Our third main contribution is the *unified treatment and generalization of all previously known safe queries*, i.e. those that can be evaluated with a query plan in polynomial time in the size of the database (Section 5). Safe conjunctive queries without self-joins have so far been defined for query reliability based on the occurrence of variables across subgoals (“hierarchical queries”), or taking into account func-

tional dependencies, notably keys [15,51], or if a subset of relations is deterministic [15]. We show that our approach naturally generalizes all safe queries: for every query that is either safe (whether due to variable co-occurrence, key constraints, or presence of deterministic tables) or just data-safe<sup>2</sup>, reliability and propagation scores always coincide; for every query that is not safe, our approach still returns a unique, well-defined score in polynomial time.

Our forth main contribution is a set targeted *multi-query optimization techniques* that considerably speed up the time needed to evaluate the propagation score (Section 6). Instead of evaluating each minimal plan separately, we merge them into one single query plan and further use views to reuse intermediate results to reduce the amount of redundant calculation. We further show how to simplify the plan in the presence of functional dependencies or a subset of relations which are deterministic.

*Outline.* We review and introduce basic definitions and notations (Section 2), then formally introduce query dissociation and the propagation score (Section 3). We prove its strong connection to query plans (Section 4), then give two optimizations based on schema knowledge (Section 5) and three general optimizations to calculate propagation very efficiently (Section 6). We evaluate our approach (Section 7), discuss related work (Section 8), before we conclude (Section 9). All proofs are included in the appendix.

*Prior publications.* Section 3, and parts of Section 4 are based on a workshop paper [29]. Section 4 includes a new, simpler minimal query enumeration algorithm. The optimizations from Section 5 and Section 6 make query evaluations actually practical and are new, as well as the extensive evaluations of Section 7. The appendix includes the complete set of proofs, plus new connections between relevance on graph and hypergraphs. In very recent work [30], 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. In this paper, we only focus on upper bounds for ranking, as we had previously shown them to be tighter than lower bounds.

## 2 Background

We consider probabilistic databases (PDBs) where each tuple  $t$  has an independent probability  $p(t) \in [0, 1]$ . We denote with  $D$  the database instance, i.e. the collection of tuples and their probabilities. We use bold notation (e.g.,  $\mathbf{x}$ ) to denote both sets or tuples,  $[k]$  to denote the set  $\{1, \dots, k\}$ , and  $\mathbf{x}_{[i,j]}$  as short form for  $(x_i, \dots, x_j)$ . A possible world is generated by independently including each tuple  $t$  in the

<sup>2</sup> A query  $q$  over a database  $D$  is “data-safe” if there exists a query plan that calculates the query reliability  $q$  over  $D$  correctly. The notion of data-safety thus generalizes the query-centric notion of safety by including the database instance (see [42] for details).

world with probability  $p(t)$ . Thus, the database  $D$  is *tuple-independent*. Consider a *self-join-free conjunctive query*  $q(\mathbf{x}) :- g_1, \dots, g_m$ , where  $g_1, \dots, g_m$  are relational atoms, also called subgoals, over a vocabulary  $R_1, \dots, R_m$ . This notation is an abbreviation for the first-order formula  $q(\mathbf{x}) = \exists x_1 \dots \exists x_k. (g_1 \wedge \dots \wedge g_m)$  where  $x_1, \dots, x_k$  are the bound variables and  $\mathbf{x}$  the free variables in the formula. “Self-join-free” implies that every subgoal  $g_i$  refers to a different relation  $R_i$ . We denote by  $\text{Var}(q)$  the set of all variables, by  $\text{HVar}(q)$  the set of *head variables*  $\mathbf{x}$ , and by  $\text{EVar}(q)$  the set of non-head or *existential variables* of a query  $q$ . For a Boolean query:  $\text{HVar}(q) = \emptyset$  and  $\text{Var}(q) = \text{EVar}(q)$ . We also write  $\text{Var}(g_i)$  for the variables in a subgoal  $g_i$ , and  $A$  for the active domain of a database  $D$ . The focus of probabilistic query evaluation is to compute  $\mathbb{P}[q]$ , which is the probability that the query is true in a randomly chosen world, and which we call the *query reliability*  $r(q)$  [36].

It is known that the data complexity [69] of any conjunctive query  $q$  is either PTIME or #P-hard [16]. The class of PTIME queries, also called *safe queries*, for the case of *self-join-free* conjunctive queries are precisely the *hierarchical queries* [15] defined as follows:

**Definition 2 (Hierarchical queries)** For every variable  $x$  in  $q$ , denote  $sg(x)$  the set of subgoals that contain  $x$ . Then  $q$  is called hierarchical iff for any two existential variables  $x, y$ , one of the following three conditions hold:  $sg(x) \subseteq sg(y)$ ,  $sg(x) \cap sg(y) = \emptyset$ , or  $sg(x) \supseteq sg(y)$ .

For example, the query  $q_1 :- R(x, y), S(y, z), T(y, z, u)$  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  $z$ . Every hierarchical query can be computed in PTIME, but non-hierarchical queries are #P-hard, in general.<sup>3</sup>

We next define *query plans* for conjunctive queries:

**Definition 3 (Query plan)** Let  $R_1, \dots, R_m$  be a relational vocabulary. A query plan is given by the grammar

$$P ::= R_i(\mathbf{x}) \mid \pi_{\mathbf{x}} P \mid \bowtie [P_1, \dots, P_k]$$

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

Note the requirement in our definition that joins and projections alternate. Hence, we do not consider permutations in the joins (called join orderings [48]). Also, we do not allow join trees such as  $\bowtie [\bowtie [R_1, R_2], R_3]$  or  $\bowtie [R_1, \bowtie [R_2, R_3]]$ . Instead, both are considered equivalent and

<sup>3</sup> Non-hierarchical queries can become safe when considering functional dependencies [15,51] or deterministic tables [15].

uniquely represented by the query plan  $\bowtie [R_1, R_2, R_3]$ , up to reordering of the operands. The reason is that those different join orderings all have the same probabilistic semantics, i.e. they result in the same query scores as defined further below. We do not focus on query optimization in this paper until Section 6.

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

$$\text{HVar}(R_i(\mathbf{x})) = \mathbf{x}$$

$$\text{HVar}(\pi_{\mathbf{x}}(P)) = \mathbf{x}$$

$$\text{HVar}(\bowtie [P_1, \dots, P_k]) = \bigcup_{i=1}^k \text{HVar}(P_i)$$

A plan is called Boolean if  $\text{HVar}(P) = \emptyset$ . We assume the usual sanity conditions on plans to be satisfied: in a project operator  $\pi_{\mathbf{x}}(P)$  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  $\pi_{\mathbf{x}}(P)$  s.t.  $y \in \text{HVar}(P) - \mathbf{x}$ . For notational convenience, we also use the “project-away” notation: Given a subplan  $P$  with head variables  $\text{HVar}(P) = \mathbf{x} \cup \mathbf{y}$ , the *project-away operator*  $\pi_{\mathbf{y}}P$  projects a subset of the variables  $\mathbf{y} \neq \emptyset$  away and onto  $\mathbf{x}$ :  $\pi_{\mathbf{y}}P \equiv \pi_{\mathbf{x}}P$ .

A plan  $P$  is evaluated on a probabilistic database  $D$  using an *extensional semantics* [27, 54]<sup>4</sup>: Each subplan  $P$  returns an intermediate relation of arity  $|\text{HVar}(P)| + 1$ . The extra *probability attribute* stores the probability of each tuple. To compute the probability, each operator assumes the input tuples to be *independent*, i.e. the *probabilistic join operator*  $\bowtie^P[\dots]$  multiplies the tuple probabilities  $\prod_{i=1}^k p_i$  for  $k$  tuples that are joined, and the *probabilistic project operator* with duplicate elimination  $\pi^P$  computes the probability as  $1 - \prod_{i=1}^k (1 - p_i)$  for  $k$  tuples with the same projection attributes [27, 58]. For a Boolean plan  $P$ , this results in a single probability value, which we denote  $\text{score}(P)$ . In general, this is not the correct query reliability  $r(q_P)$ , which, recall, is defined in terms of possible worlds:  $\text{score}(P) \neq r(q_P)$ .

**Definition 4 (Safe plan)** A plan  $P$  is called safe if for any join operator  $\bowtie^P[P_1, \dots, P_k]$  all subplans have the same head variables:  $\text{HVar}(P_i) = \text{HVar}(P_j)$ , for all  $1 \leq i, j \leq k$ .

The following are known facts about the relation between safe queries and safe plans [15].

**Proposition 5 (Safety)** Let  $P$  be a plan for the conjunctive self-join-free query  $q$ . Then: (1)  $\text{score}(P) = r(q)$  for any probabilistic database iff  $P$  is safe; (2)  $q$  admits a safe plan iff  $q$  is hierarchical. Moreover, the safe plan is unique (up to permutation in the join orders).

<sup>4</sup> *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.

*Example 6 (Safe plans and SQL)* We illustrate safe plans of hierarchical queries with two examples, starting with a simple hierarchical query, its unique safe plan, and its translation into SQL assuming the schema  $R(A, B), S(A)$  and each table having one additional attribute  $P$  for the probability of a tuple. Here  $\text{IOR}(\mathbf{x})$  is a user-defined aggregate that calculates the independent-or for the probabilities of grouped tuples, i.e.  $\text{IOR}(p_1, p_2, \dots, p_n) = 1 - (1 - p_1)(1 - p_2) \cdots (1 - p_n)$  (the exact UDA definition for PostgreSQL is stated in [30]).

$$q_1(x) :- R(x, y), S(x)$$

$$P_1 = \bowtie^P [\pi_{\mathbf{y}}^P R(x, y), S(x)]$$

```
select R2.A, R2.P * S.P as P
from (select A, IOR(P) as P
      from R
      group by A) as R2, S
where R2.A = S.A
```

The second one is a slightly more complicated Boolean query:

$$q_2 :- R(x, y), S(y, z), T(y, z, u)$$

$$P_2 = \pi_{\mathbf{y}}^P \bowtie^P [\pi_{\mathbf{x}}^P R(x, y), \pi_{\mathbf{z}}^P \bowtie^P [S(y, z), \pi_{\mathbf{u}}^P T(y, z, u)]] \quad \blacksquare$$

### 3 Dissociation and Propagation

In this section, we define our technique of *query dissociation* and the *propagation score* of a query. We first define the approach formally, then describe in the following sections efficient methods to evaluate query dissociation and propagation. We commonly first give formal definitions, then provide the intuition with examples later.

**Definition 7 (Query dissociation)** A *dissociation* of a conjunctive query  $q :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$  is a collection of sets of variables  $\Delta = (\mathbf{y}_1, \dots, \mathbf{y}_m)$  with  $\mathbf{y}_i \subseteq \text{Var}(q) - \text{Var}(g_i)$ . The dissociated query is:

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

Thus syntactically, query dissociation adds to some subgoals, variables of other subgoals. The natural abstraction to understand query dissociation is with the help of the *incidence matrix* of a query.

**Definition 8 (Incidence matrix)** The *incidence matrix* of query  $q$  includes one row for each subgoal and one column for each variable in  $\text{Var}(q)$ . An empty circle in a particular row and column indicates that the corresponding subgoal contains the corresponding variable. A full circle indicates that the corresponding subgoal is dissociated on the corresponding variable.

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

**Definition 9 (Table dissociation)** Given a conjunctive query  $q: -R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ , the active domain  $A$ , and a query dissociation  $\Delta = (\mathbf{y}_1, \dots, \mathbf{y}_m)$ . The *dissociation of table  $R_i$*  on  $\mathbf{y}_i = \{y_{i1}, \dots, y_{ik}\}$  is the relation given by query

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

Together, both definitions define the semantics of query dissociation as follows: Add some 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. Note that this is the semantics of a dissociated query, and not the way we actually evaluate queries. In the following sections we describe methods that evaluate the dissociated query without modifying the tables in the database.

*Example 10 (Query dissociation)* We illustrate with the query  $q: -R(x), S(x, y), T(y)$ , the dissociation  $\Delta = (\{y\}, \emptyset, \emptyset)$ , and the database instance in Fig. 3c. The active domain  $A$  is  $\{a, b, c\}$ , and a variable  $p_i$  stands for the independent probability of a tuple with index  $i$ . The resulting dissociated query is  $q^\Delta: -R^{\{y\}}(x, y), S^0(x, y), T^0(y)$ , which we also abbreviate as  $q^\Delta: -\tilde{R}(x, \tilde{y}), S(x, y), T(y)$ . Thus, we use the tilde ( $\tilde{\cdot}$ ) as a short notation to indicate dissociated relations and variables, e.g.,  $\tilde{R}(x, \tilde{y})$  for the dissociation of  $R(x)$  on  $y$ . Fig. 3d shows the new database instance with table  $\tilde{R}$  as the original table  $R$  dissociated on variable  $y$ . Note that the original tuple  $R(a)$  got dissociated into two tuples  $R(a, b)$  and  $R(a, c)$  with the same probability  $p_1$ <sup>5</sup>. Fig. 3a and Fig. 3b show the incidence matrices of both  $q$  and its dissociation  $q^\Delta$ . Finally, note the similarity of this example to the example from the introduction (Example 1 and Fig. 1). ■

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

<sup>5</sup> A third tuple  $R(a, a)$  is not shown since it does not join with any other tuple in  $q$ . Hence, a semi-join reduction on the input tables can be applied without changing the semantics of dissociation.

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

**Fig. 3** Example 10. Incidence matrices of query  $q: -R(x), S(x, y), T(y)$  and dissociation  $q^\Delta: -\tilde{R}(x, \tilde{y}), S(x, y), T(y)$ . Original database instance  $D$  and new instance  $D^\Delta$  with table  $R$  dissociated on variable  $y$ .

**Definition 11 (Partial dissociation order)** We define the partial order on the dissociations of a query as:

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

**Theorem 12 (Partial dissociation order)** For every two dissociations  $\Delta$  and  $\Delta'$  of a query  $q$ , the following holds over every database instance:

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

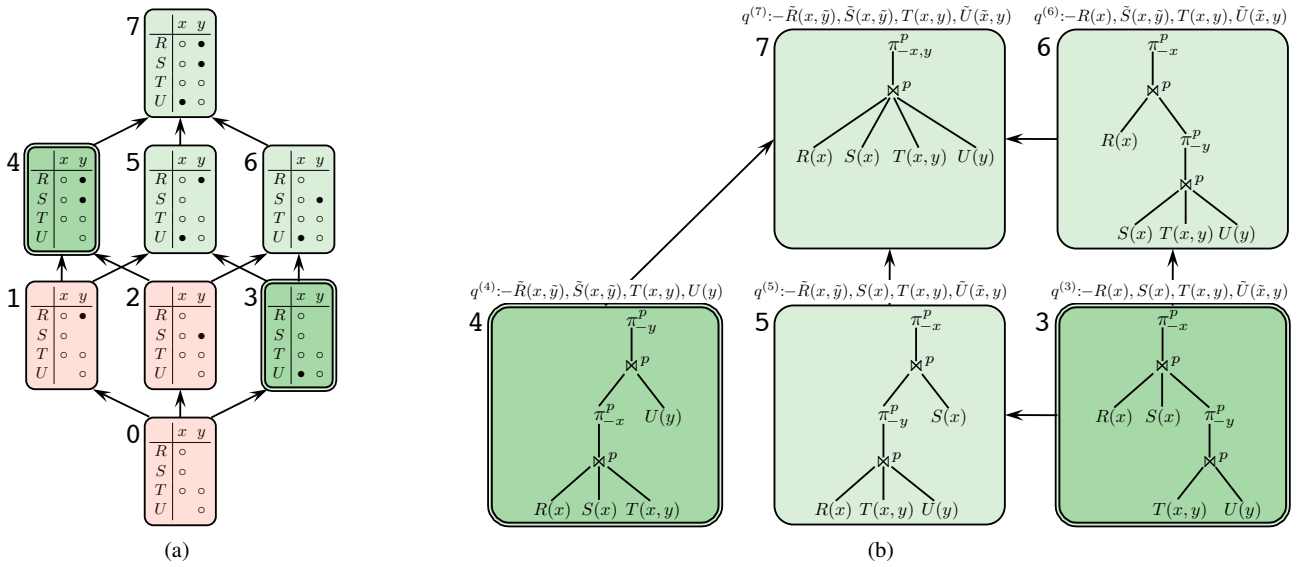
**Corollary 13 (Upper query bounds)** For every database and every dissociation  $\Delta$  of a query  $q: r(q^\Delta) \geq r(q)$ .

Corollary 13 immediately follows from Theorem 12 as every query is a dissociation of itself on the collection of empty sets. The total number of dissociations corresponds to the cardinality of the power set of variables that can be added to subgoals. Hence, for every query with  $n$  non-head variables and  $m$  subgoals, there are  $2^{K|}$  possible dissociations with  $K = \sum_{i=1}^m (n - |\text{Var}(g_i)|)$  forming a partial order in the shape of a power set (see Fig. 4a for Example 16). Notice that the dissociation scores  $r(q^\Delta)$  can be evaluated efficiently for the subset of dissociations that are safe:

**Definition 14 (Safe dissociation)** A dissociation  $\Delta$  of a query  $q$  is called *safe* if the dissociated query  $q^\Delta$  is safe.

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

Recall at this point, that the motivation of this work is to develop an efficient and *unique* semantics analogous to propagation on directed graphs. Queries have no concept of direction, and we suggest that the *choice of direction* in a graph corresponds to a particular *choice of dissociation* for a query that makes the query safe. To get uniqueness independent of a particular dissociation, we suggest in this paper the *propagation score*  $\rho(q)$  of a query as the minimum probability of all those dissociations in the partial dissociation order of a query that are *safe*, i.e. those which admit a safe plan given the dissociated tables.



**Fig. 4** Example 16. (a): Partial dissociation order for  $q: -R(x), S(x), T(x, y), U(y)$ . Safe dissociations are shaded in green (3 to 7), *minimal safe dissociations* in dark green (3 and 4). (b): All 5 possible query plans for  $q$ , their partial order and the correspondence to safe dissociations (3 to 7).

**Definition 15 (Propagation)** The *propagation score*  $\rho(q)$  for a query  $q$  is the minimum score of all safe dissociations:  $\rho(q) = \min_{\Delta} r(q^{\Delta})$ , where  $\Delta$  ranges over safe dissociations.

*Example 16 (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 shown in Fig. 4a. Of those 8 dissociations, five are safe and shaded in green. Furthermore, two of those safe dissociations are minimal and emphasized in dark green:

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

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

To illustrate that these plans are upper bounds, consider a database with  $S = \{(1, 1), (1, 2), (2, 2)\}$ ,  $R = T = U = \{1, 2\}$ , and all tuples having probability  $p = 1/2$ . Then  $q$  has probability  $83/2^9 \approx 0.161$ , while the  $q^{(3)}$  has probability  $169/2^{10} \approx 0.165$ , and  $q^{(4)}$  has probability  $353/2^{11} \approx 0.172$ , both of which are upper bounds. The propagation score is the minimum score of all minimal safe dissociations, and thus 0.165:  $\rho(q) = \min_{i \in \{3, 4\}} [r(q^{(i)})]$ . ■

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 Corollary 13. Hence, query propagation 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.

Also recall that our original motivation was to develop for queries a concept that is analogous to propagation on directed networks. 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<sup>6</sup>. Further, the propagation score over this network corresponds to one of several possible dissociations of this query  $q$ . Thus, *query dissociation is a strict generalization of network propagation on  $k$ -partite graphs*. And we define query propagation as corresponding to a certain dissociation with minimum reliability (see Fig. 5).

**Proposition 17 (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  $r(q)$  with:

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

<sup>6</sup> 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. The expression *chain query* derives from the observation that its (hyper)graph resembles a simple chain.

Networks / Graphs	Conjunctive queries
<b>Network reliability:</b> Probability that two nodes are connected. <i>Independent</i> of edge direction.	<b>Query reliability:</b> Probability that query is true in a random world. <i>Independent</i> of query plan.
<b>Propagation score:</b> ‘Relatedness’ propagates from source to target. <i>Dependent</i> on edge direction. <i>Upper bound</i> to reliability.	<b>Dissociation score:</b> A query plan evaluates from leafs to root. <i>Dependent</i> on choice of dissociation. <i>Upper bound</i> to reliability.
	<b>Propagation score:</b> <i>Minimum</i> over all possible safe dissociations. <i>Unique</i> for given query.

**Fig. 5** Connection between reliability and propagation in networks and conjunctive queries: In contrast to the propagation score on networks, the propagation score for CQs is the minimum over all possible dissociations, and is therefore unique for every query and database.

- (b) *The directed propagation score from  $s$  to  $t$  (as defined in Example 1) is  $r(q^\Delta)$  with:*

$$q^\Delta :- R_1^{\mathbf{x}_{[3,k]}}(s, \mathbf{x}_{[2,k]}), R_2^{\mathbf{x}_{[4,k]}}(\mathbf{x}_{[2,k]}), \dots, R_k^0(x_k, t)$$

#### 4 Dissociations and Plans

So far, in order to compute the propagation score of a query  $q$ , we need to dissociate its tables and compute several dissociated safe queries  $q^\Delta$ . In practice, we will not apply naively query dissociation (Definition 7) because the table dissociation part (Definition 9) computes several cartesian products and is very inefficient. Our second technical result is a deep connection between safe dissociations and query plans, which allows us to perform dissociation very efficiently.

**Theorem 18 (Safe dissociation)** *Let  $q$  be a conjunctive query without self-joins. There exists an isomorphism between safe dissociations  $\Delta$  of  $q$  and query plans  $P$  for  $q$ . Moreover, the reliability of the dissociated query is equal to the score of the corresponding plan:  $r(q^\Delta) = \text{score}(P^\Delta)$ .*

We next describe this *isomorphism*. Consider a safe dissociation  $q^\Delta$  and denote its corresponding unique safe plan  $P^\Delta$ . This plan uses dissociated relations, hence each relation  $R_i^{\mathbf{y}_i}(\mathbf{x}_i, \mathbf{y}_i)$  has some extraneous variables  $\mathbf{y}_i$ . Drop all the variables  $\mathbf{y}_i$  from the relations, and from all operators that use them: this transforms  $P^\Delta$  into a regular, generally unsafe plan  $P$  for  $q$ . Conversely, consider any plan  $P$  for  $q$ . We define its corresponding safe dissociation  $\Delta$  as follows. For each join operation  $\bowtie^p [P_1, \dots, P_k]$ , let its *join variables*  $\text{JVar}$  be the union of the head variables of all subplans:  $\text{JVar} = \bigcup_j \text{HVar}(P_j)$ . For every relation  $R_i$  occurring in  $P_j$ , add the missing variables  $\text{JVar} - \text{HVar}(P_j)$  to  $\mathbf{y}_i$ . For example, consider the lower join in query plan 5 of Fig. 4b:

$$\bowtie^p [R(x), T(x, y), U(y)]$$

Here,  $\text{JVar} = \{x, y\}$ , and the corresponding safe dissociation of this subplan is

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

The complete Boolean query  $q^{(5)}$  is shown at the top of query plan 5. Note that while there is a one-to-one mapping between safe dissociations and query plans, unsafe dissociations do not correspond to plans. For example, dissociations 0 to 2 in Fig. 4a are still hard.

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

**Corollary 19 (Query plans as upper bounds)** *Let  $P$  be any plan for a query  $q$ . Then  $\text{score}(P) \geq r(q)$ .*

In other words, any query plan as defined in Definition 3 always gives an upper bound to the actual query reliability  $r(q)$ . To the best of our knowledge, this common property of query plans was only previously mentioned in [15], but has never been proven.

To summarize, Theorem 18 gives us a much more efficient method for computing the propagation score  $\rho(q)$  of a query: iterate over all plans  $P$ , compute their scores, and retain the minimum score  $\min_P [\text{score}(P)]$ . Each plan  $P$  is evaluated directly on the original probabilistic database, and there no need to dissociate the input tables. However, this approach is *still inefficient* as it computes some ‘redundant’ plans: for example, in Fig. 4 plans 5, 6, 7 are redundant as they are all greater than plan 3 in the partial dissociation order. We also say that plan 3 *dominates* plans 5, 6, and 7. It thus suffices to evaluate only the *minimal query plans*, i.e. those for which the corresponding dissociation is minimal (or not dominated) among all safe dissociations: in our Example 16, these are plans 3 and 4.

**Example 20 (Query Plans)** We saw in Example 16 that the query  $q :- R(x), S(x), T(x, y), U(y)$  has 8 dissociations depicted in Fig. 4a. Among those, five are safe, and Fig. 4b shows their corresponding query plans. The two dissociations  $q^{(3)}$  and  $q^{(4)}$  are both *safe and minimal*, and their corresponding query plans over the original tables are:

$$P^{(3)} = \pi_{-x}^p \bowtie^p [R(x), S(x), \pi_{-y}^p \bowtie^p [T(x, y), U(y)]]$$

$$P^{(4)} = \pi_{-y}^p \bowtie^p [U(y), \pi_{-x}^p \bowtie^p [R(x), S(x), T(x, y)]]$$

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

We next describe our third technical result, the recursive Algorithm 1 that generates all minimal plans for a given query  $q$ . We start with some necessary notions: We



```

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

```

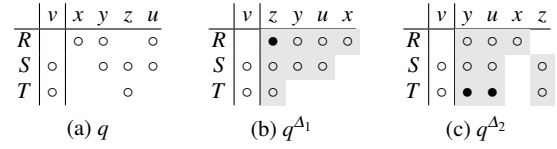
**Algorithm 1** generates all minimal query plans for a given query  $q$ .

call a variable  $x$  a *root variable* of a query plan  $P$  and write  $x \in \text{RVar}(P)$  if  $x$  is projected away in the last operation:  $P = \pi_x^p P'$  and  $x \in \mathbf{x}$ . We say a query is *connected* if all subgoals are connected by considering only existential variables  $\text{EVar}(q)$ <sup>7</sup>. For a connected query  $q$ , we write  $\text{TopSets}(q)$  for the set of minimal subsets of existential variables, for which removing them disconnects the query. We call each such set of variables a *top set*. Minimal implies that no proper subset  $\mathbf{y}' \subset \mathbf{y} \in \text{TopSets}(q)$  can disconnect the query. The algorithm works as follows: If the query has one subgoal (line 1), then we only need to project on the head variables.<sup>8</sup> If the query has more than one subgoal, then we first see whether the query is connected. When the query is disconnected (line 3), then the algorithm recursively computes the minimal query subplans for each connected component, then creates a query plan for each combination of those subplans. When the query is connected (line 7), the algorithm creates a separate plan for each top set of variables  $\mathbf{y} \in \text{TopSets}(q)$  by moving them from the existential variables  $\text{EVar}(q)$  to the head variables  $\text{HVar}(q)$ , thereby disconnecting the query. In sum, the algorithm recursively alternates between the two steps of independent projection and independent join until reaching a single subgoal. And at each independent project, the algorithm considers all possible minimal subplans.

**Example 21 (Enumerate minimal query plans)** Consider the non-Boolean query  $q(v) :- R(x, y, u), S(y, z, u, v), T(z, v)$ . Fig. 6a shows its incidence matrix with its variables separated into the head variable  $\text{HVar}(q) = \{v\}$  and existential variables  $\text{EVar}(q) = \{x, y, z, u\}$ . The query is connected, and

<sup>7</sup> 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 connected components of  $q'$  by any variable. The query is connected if  $k = 1$ , otherwise it is disconnected, and  $\forall i \neq j : \text{VVar}(q_i) \cap \text{VVar}(q_j) \subseteq \text{HVar}(q)$ .

<sup>8</sup> Note that if there are no existential variables ( $\mathbf{x} = \mathbf{x}_i$ ), then there is no need for the projection operator and one could instead simplify to  $\mathcal{P} \leftarrow \{R_i(\mathbf{x}_i)\}$ , instead of  $\mathcal{P} \leftarrow \{\pi_x^p R_i(\mathbf{x}_i)\}$ .



**Fig. 6** Example 21. Query  $q$  and its two minimal safe dissociations. Note the hierarchy between  $\text{EVar}(q)$  for each safe dissociation.

out of  $2^4$  different subsets of  $\text{EVar}(q)$ , there are 2 top sets, i.e. minimum sets of variables which after removing them disconnects the query:  $\text{TopSets}(q) = \{\{z\}, \{y, u\}\}$ . Projecting on the first top set  $\{z\}$  allows to separate the query into  $q_1(z, v) :- R(x, y, u), S(y, z, u, v)$  and  $q_2(z, v) :- T(z, v)$ . Note that  $q_1$  and  $q_2$  have no existential variables in common. Otherwise stated, they share only head variables  $z$  and  $v$ . Projecting on the top set  $\{y, u\}$  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:

$$P^{(1)} = \pi_z^p \bowtie^p [\pi_{y,u}^p \bowtie^p [\pi_x^p R(x, y, u), S(y, z, u, v)], T(z, v)]$$

$$P^{(2)} = \pi_{y,u}^p \bowtie^p [\pi_x^p R(x, y, u), \pi_z^p \bowtie^p [S(y, z, u, v), T(z, v)]]$$

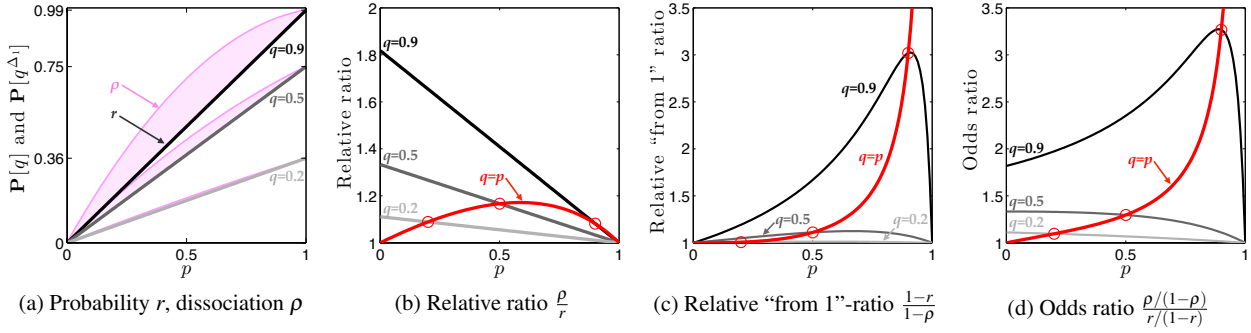
Fig. 6b and Fig. 6c show their respective safe dissociations, where the existential variables are ordered so that the hierarchy implied by the query plans can be seen. ■

**Theorem 22 (Algorithm 1)** *Algorithm 1 returns a sound and complete enumeration of minimal query plans. It 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.*

We next comment on approximation quality. It is known that approximating probabilistic reasoning is NP-hard, and thus deterministic approximation scheme can not give general approximation guarantees [61]. However, it is interesting to note that the *approximation quality of dissociation increases as the input probabilities decrease*. The practical implication is that the rankings returned by dissociation are better if the input probabilities are small.

**Proposition 23 (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 query reliability  $r(q)$  by the propagation score  $\rho(q)$  decreases as  $f$  goes to 0:  $\lim_{f \rightarrow 0^+} \frac{\rho(q) - r(q)}{r(q)} \rightarrow 0$ .*

In the following analytic example, we illustrate Prop. 23 by calculating the relative ratio between propagation and reliability for changing input probabilities.



**Fig. 7** Example 24: Comparing the probabilities  $r := \mathbb{P}[q]$  and  $\rho := \mathbb{P}[q^A]$  for varying input probabilities.

*Example 24 (Small probabilities)* We consider the Boolean query  $q: -R(x), S^d(x, y), T(y, z)$  and the dissociation  $q^A: -\bar{R}(x, \bar{y}), S^d(x, y), T(y, z)$  over the database instance  $r_1 = R(a), s_1 = S^d(a, b), s_2 = S^d(a, c), t_1 = T(b),$  and  $t_2 = T(c)$ . With deterministic relation  $S^d$ , the lineages of  $q$  and  $q^A$  are  $\text{Lin}(q) = r_1 t_1 \vee r_1 t_2$  and  $\text{Lin}(q^A) = r_1 t_1 \vee r_1' t_2$ , respectively. Assuming  $\mathbb{P}[r_1] = p$  and  $\mathbb{P}[t_1] = \mathbb{P}[t_2] = q$ , the respective probabilities become  $r := \mathbb{P}[q] = p(1 - q^2) = pq(2 - q)$  and  $\rho := \mathbb{P}[q^A] = 1 - (1 - pq)^2 = pq(2 - pq)$ . There are four justifiable metrics to measure the approximation quality of dissociation  $\rho$  with regard to the actual probability  $r$ : (1) their *absolute difference*  $\rho - r$ , which is not meaningful when both are too close to either 0 or 1; (2) their *relative ratio*  $\frac{\rho}{r}$ , which is not meaningful close to 1; (3) their *relative "from 1"-ratio*  $\frac{1-r}{1-p}$ , which is not meaningful close to 0; or (4) the *odds ratio*  $\frac{\rho/(1-p)}{r/(1-r)}$ , 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  $\geq 1$ . Also notice that the relative error  $\frac{\rho-r}{r}$  is equal to the relative ratio  $\frac{\rho}{r}$  minus 1.

Fig. 7a shows the original probabilities  $r$  (full lines) and those of their dissociations  $\rho$  (border of shaded areas) for various values of  $p$  and  $q$ . 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. Fig. 7b, Fig. 7c, and Fig. 7d show the approximation quality in terms of our three previously defined ratios. Notice that the red line varies both  $p$  and  $q$  at the same time by keeping  $p = q$ . 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 gets close to 1. ■

We end this section by commenting on the number of minimal safe dissociations. Not surprisingly, this number is exponential in the size of the query. To see this, consider a Boolean  $k$ -star query<sup>9</sup>  $q: -R_1(x_1), \dots, R_k(x_k), U(x_1, \dots, x_k)$ . There are exactly  $k!$  minimal safe dissociations: Take any

<sup>9</sup> 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

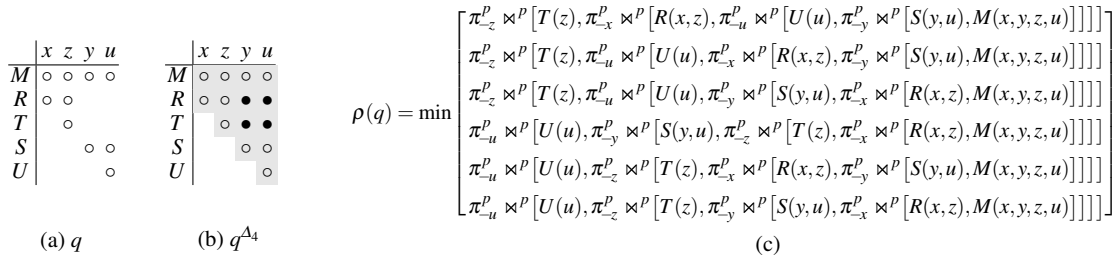
$k$ -star query				$k$ -chain query			
$k$	#MP	#P	# $\Delta$	$k$	#MP	#P	# $\Delta$
1	1	1	1	2	1	1	1
2	2	3	4	3	2	3	4
3	6	13	64	4	5	11	64
4	24	75	4096	5	14	45	4096
5	120	541	$> 10^6$	6	42	197	$> 10^6$
6	720	4683	$> 10^9$	7	132	903	$> 10^9$
7	5040	47293	$> 10^{12}$	8	429	4279	$> 10^{12}$
seq	$k!$	A000670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

**Fig. 8** Number of minimal plans, total plans, and total dissociations for star and chain queries. A stands for the corresponding OEIS sequence.

consistent preorder  $\preceq$  on the variables. It must be a total preorder, i.e. for any  $i, j$ , either  $x_i \preceq x_j$  or  $x_j \preceq x_i$ , because  $x_i, x_j$  occur together in  $U$ . Since it is minimal,  $\preceq$  must be an order, i.e. we can't have both  $x_i \preceq x_j$  and  $x_j \preceq x_i$  for  $i \neq j$ . Therefore,  $\preceq$  is a total order, and there are  $k!$  such. Note that while the number of safe 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 [69] for all queries. Figure 8 gives an overview of the number of minimal query plans, total query plans, and dissociations for star and chain queries. Appendix H contains the derivation. Recall that in our definition of query plans, we *do not consider permutations in the joins* (called join orderings [48]). 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.

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 instance al-

fact that each variable appears in exactly two relations implies that the dual hypergraph is actually a standard graph. The expression *star query* derives from the observation that its dual (hyper)graph resembles a star with the table  $U$  connected to all other relations.



**Fig. 9** Example 25. Incidence matrix (a) for our running example  $q: -R(x,z), S(y,u), T(z), U(u), M(x,y,z,u)$  together with (b) one minimal safe dissociation. (c): All 6 minimal query plans generated by algorithm 1. The propagation score is the minimum of the scores of those plans.

lows 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 to leverage schema knowledge to simplify and reduce the number of minimal query plans. We consider *deterministic relations* (i.e. relations with all tuple having probabilities of 1), and *functional dependencies* (e.g., keys). We will use the following running example throughout this and the next section:

*Example 25 (Running example)* Consider the query  $q$ :

$$q: -R(x,z), S(y,u), T(z), U(u), M(x,y,z,u)$$

Our default is to evaluate all minimal plans returned by Algorithm 1, then take the minimum score. Figure 9 shows the incidence matrix of  $q$  together with the 6 minimal plans. ■

### 5.1 Deterministic relations

We denote deterministic tables with an exponent, i.e. a table  $R$  is probabilistic, and a table  $R^d$  is deterministic. We call *separator variables* of a query  $q$  the set of existential variables which appear in all probabilistic subgoals and denote them as  $SVar(q)$ . We start with the following lemma:

**Lemma 26 (Deterministic table dissociation)** *Dissociating a deterministic table in a query  $q$  on any variable does not affect the query reliability:  $r(q^{A_4}) = r(q)$ .*

In the following, whenever we dissociate a deterministic relation, we mark the dissociated variables in the incidence matrix with a star ( $\star$ ) instead of a bullet ( $\bullet$ ) to emphasize that this dissociation does not change the query reliability. Lemma 26 seems to suggest that we could dissociate all deterministic relations and then apply our standard algorithm to find the set of minimal plans of a query. However, this is not correct as we illustrate with a counter-example:

**Recursive algorithm: DP (PlansWithDeterministicTables)**  
**Input:** Query  $q(x) :- R_1(x_1), \dots, R_m(x_m)$   
 Schema information on deterministic relations  
**Output:** Set of all minimal query plans  $\mathcal{P}$

```

1 if  $m = 1$  then  $\mathcal{P} \leftarrow \{\pi_x^p R_i(x_i)\}$  else
2   Set  $\mathcal{P} \leftarrow \emptyset$ 
3   if  $q$  is disconnected then
4     Let  $q = q_1, \dots, q_k$  be the components connected by  $EVar(q)$ 
5     foreach  $q_i$  do Let  $HVar(q_i) \leftarrow HVar(q) \cap VAr(q_i)$  foreach
6        $(P_1, \dots, P_k) \in DP(q_1) \times \dots \times DP(q_k)$  do
7          $\mathcal{P} \leftarrow \mathcal{P} \cup \{\bowtie^p [P_1, \dots, P_k]\}$ 
8   else
9     if there are separator variables  $\mathbf{z} = SVar(q) \neq \emptyset$  then
10      Let  $q' \leftarrow q$  with  $HVar(q') \leftarrow HVar(q) \cup \mathbf{z}$ 
11      if  $q'$  is disconnected then
12        foreach  $P \in DP(q')$  do  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\pi_{\mathbf{z}}^p P\}$ 
13      else
14        foreach  $\mathbf{y} \in TopSets(q')$  do
15          Let  $q'' \leftarrow q$  with  $HVar(q'') \leftarrow HVar(q) \cup \mathbf{z} \cup \mathbf{y}$ 
16          foreach  $P \in DP(q'')$  do  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\pi_{(\mathbf{z}, \mathbf{y})}^p P\}$ 
17      else
18        foreach  $\mathbf{y} \in TopSets(q)$  do
19          Let  $q' \leftarrow q$  with  $HVar(q') \leftarrow HVar(q) \cup \mathbf{y}$ 
20          foreach  $P \in DP(q')$  do  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\pi_{\mathbf{y}}^p P\}$ 

```

**Algorithm 2** generates all minimal query plans for a query  $q$  with deterministic relations. Lines 8 to 16 extend Algorithm 1.

*Example 27 (Incorrect deterministic dissociation)* Consider the query  $q: -R(x), S(x,y), T^d(y,z), U(z)$  with deterministic relation  $T^d$  over the database instance  $D$  from Fig. 10a. Here, indexed small letters refer to tuples in the respective relations, e.g.,  $r_1$  for tuple  $R(a)$ . The lineage of  $q$  is  $Lin(q) = r_1 s_1 t_1 u_1 \vee r_1 s_1 t_2 u_2 \vee r_2 s_2 t_1 u_1 \vee r_2 s_2 t_2 u_2$ . Replacing  $t_1$  and  $t_2$  with 1, the lineage can be factored into  $Lin(q) = ((r_1 s_1) \vee (r_2 s_2))(u_1 \vee u_2)$ , which is a read-once formula. Assuming all non-deterministic tuples to have the same probability 0.5, the query reliability becomes  $\mathbb{P}[q] = 21/64 \approx 0.328$ . Query  $q$  has three top sets  $TopSets(q) = \{\{x\}, \{y\}, \{z\}\}$ , as easily seen from its incidence matrix (Fig. 10b). Dissociating  $q$  on  $y$  (Fig. 10c) results in a safe dissociation  $q^{A_1}$  that turns out to have, on the particular database instance  $D$ , exactly the same lineage expression as the original query:  $Lin(q^{A_1}) = Lin(q)$ . Since it is a safe dissociation, there is a plan  $P(q^{A_1})$  that calculates the reliability of  $q^{A_1}$  and its score is exactly the reliability of the original query:  $score(P(q^{A_1})) = r(q) \approx$

$M$	$x$	$z$	$y$	$u$
$R^d$	○	○	○	○
$T^d$	○			
$S$		○	○	
$U$		○		

$M$	$x$	$z$	$y$	$u$
$R^d$	○	○	○	○
$T^d$	○	○	★	★
$S$	★	○	★	★
$U$		○	○	○

$$\rho(q_d) = \pi_{x,u}^p \bowtie^p [U(u), \pi_{x,y}^p \bowtie^p [S(y,u), \pi_{x,z}^p \bowtie^p [T^d(z), R^d(x,z), M(x,y,z,u)]]]]$$

(c)

(a)  $q_d$ 
(b)  $q'_d$

**Fig. 11** Example 30: Knowing that relations  $R^d$  and  $T^d$  in Example 25 are deterministic allows us to selectively dissociate and thereby reduce the number of minimal plans. In this case,  $q'_d$  becomes hierarchical. Thus, Algorithm 2 returns one single minimal plan and  $\rho(q_d) = r(q_d)$ .

$R$	$A$	$S$	$A$	$T^d$	$B$	$U$	$C$
$r_1$	$a$	$s_1$	$a$	$t_1$	$c$	$u_1$	$e$
$r_2$	$b$	$s_2$	$b$	$t_2$	$c$	$u_2$	$f$

(a)  $D$

$R$	$x$	$S$	$y$	$T^d$	$z$
$r_1$	○	$s_1$	○	$t_1$	○
$r_2$	○	$s_2$	○	$t_2$	○

(b)  $q$ 
(c)  $q^{A1}$ 
(d)  $q^{A2}$

**Fig. 10** Example 27. Query  $q: -R(x), S(x,y), T^d(y,z), U(z)$  and minimal safe dissociation  $q^{A1}$  for which  $\rho(q^{A1}) = r(q)$  happens to hold on  $D$ . Dissociation  $q^{A2}$  replaces  $T^d(y,z)$  with  $\tilde{T}^d(\bar{x},y,z)$ , and thus  $r(q^{A2}) = r(q)$ . However,  $q^{A2}$  is not safe, and any further minimal safe dissociation increases the score on  $D$ . Therefore,  $\rho(q^{A2}) \geq \rho(q^{A1})$ .

0.328. However, if we had first dissociated the deterministic relation  $T^d$  into  $\tilde{T}^d(\bar{x},y,z)$  ( $q^{A2}$  in Fig. 10d), we would have a dissociated query  $q^{A2}$  that, despite having the same reliability as the original query  $r(q^{A2}) = r(q)$ , is *not safe*. Thus, in order to calculate its propagation score, we have to find the minimum query reliability over all safe dissociations of  $q^{A2}$ , each of which requires at least two more dissociations. This turns out to increase the propagation score to  $\rho(q^{A2}) = 87/256 \approx 0.340 > 0.328 \approx \rho(q^{A1}) = r(q)$ . ■

**Lemma 28 (Separator variables)** *If  $q$  is connected by its existential variables and  $x \in \text{SVar}(q)$ , then all minimal query plans have  $x$  as root variable.*

Lemma 28 gives us immediately Algorithm 2 which extends Algorithm 1 with lines 8 to 16: whenever we have separator variables  $\mathbf{z}$  at a projection, then we verify if they disconnect the query (line 10). If they alone don't disconnect it (line 13), then we iterate over all minimum variable sets that include  $\mathbf{z}$  (i.e.  $\mathbf{z} \cup \mathbf{y}$ ) and that disconnect the query.

**Theorem 29 (Algorithm 2)** *Algorithm 2 returns a sound and complete enumeration of minimal query plans in the presence of deterministic relations.*

Note that as before, if the query is safe, the algorithm will produce one single query plan. Further note that, if all relations are deterministic, then the single minimal query plan consists of one multi-join between all relations followed by a single projection:  $\pi_{\mathbf{x}} \bowtie [R(\mathbf{x}_1), \dots, R(\mathbf{x}_m)]$ . 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, *Algorithm 2 conservatively extends deterministic SQL queries to probabilistic SQL queries* in that fully deterministic queries are evaluated exactly in the same way.

*Example 30 (Running example continued)* Assume relations  $R$  and  $T$  from Example 25 to be deterministic: Algorithm 2 then returns one single minimal plan (Fig. 11c), which implies that the query is safe and our plan returns the exact probabilities. ■

We end with a short comment on actual implementation. In practice, deterministic relations do not have a probabilistic attribute, which simplifies the calculations. Consider the subplan  $P = \bowtie^p [T^d(z), R^d(x,z), M(x,y,z,u)]$  in Fig. 11c. This subquery 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

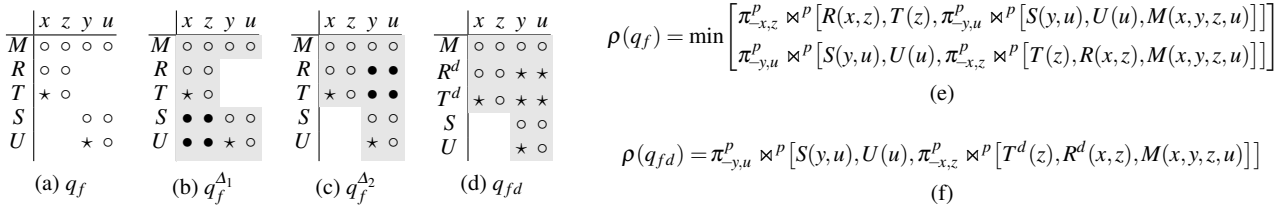
Functional dependencies (FDs), such as keys, can also reduce the number of minimal safe dissociations and thus simplify the resulting query plans. The main insight lies in the following two lemmas.

**Lemma 31 (FD dissociation and reliability)** *Given a query  $q$  with a functional dependency  $\Gamma : \mathbf{x} \rightarrow \mathbf{y}$  on relation  $R_i$ , and another relation  $R_j$  with  $\mathbf{x} \subseteq \text{Var}(R_j)$ , but  $\mathbf{y} \notin \text{Var}(R_j)$ . Then dissociating  $R_j$  on  $\mathbf{y}$  does not change the query reliability.*

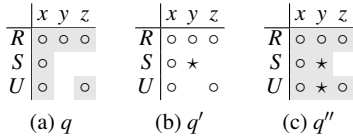
This lemma is similar to Lemma 26, and we will mark variables in the incidence matrix that are dissociated as result of an FD also with a star (★) instead of a bullet (●).

**Lemma 32 (FD dissociation and hierarchies)** *Given a safe query  $q$  and a functional dependency  $\Gamma : \mathbf{x} \rightarrow \mathbf{y}$ . Dissociating all relations  $R_j$  with  $\mathbf{x} \subseteq \text{Var}(R_j)$  on all dependent variables  $\mathbf{y} \setminus \text{Var}(R_j)$  results in a dissociation that is still safe.*

The second lemma has no analogy for deterministic relations. Recall from Example 27 and Fig. 10c that eagerly dissociating all deterministic tables in the safe dissociation  $q^{A1}$



**Fig. 12** Example 37: The query  $q_f$  (a) from our running example Example 25, after assuming two FDs  $z \rightarrow x$  and  $u \rightarrow y$ , has two minimal safe dissociations (b & c), and two corresponding minimal query plans returned by Algorithm 3 (e). If we know, in addition, that  $R^d$  and  $T^d$  are deterministic, then the query becomes safe (d) and has one single minimal query plan (f).



**Fig. 13** Example 33. Applying an FD  $x \rightarrow y$  only to a subset of relations can turn the previously safe query  $q$  into an unsafe  $q'$ . However,  $q$  cannot become unsafe when applying  $\Gamma$  to all relation (here  $q''$ ).

would lead to an unsafe dissociation. With a functional dependency  $\Gamma : \mathbf{x} \rightarrow \mathbf{y}$ , however, we can first eagerly dissociate the dependent variables  $\mathbf{y}$  in *all* relations that include  $\mathbf{x}$  as variables, then apply our previous algorithm. Note the emphasis on *all* relations as this property does not hold for arbitrary subsets as we illustrate again with a counter-example.

**Example 33** (Incorrect FD dissociation) Consider the query  $q : -R(x,y,z), S(x), T(x,z)$  with functional dependency  $\Gamma : x \rightarrow y$  holding in relation  $R$ . The query is safe (the hierarchy is shown in gray in Fig. 13a) and has one single plan  $P = \pi_{-x}^p \bowtie^p [S(x), \pi_{-z}^p \bowtie^p [\pi_{-y}^p R(x,y,z), U(x,z)]]$ . Dissociating relation  $S$  on  $y$  does not change the reliability, however it makes the query unsafe ( $q'$  in Fig. 13b) with now two minimal plans. If we instead dissociate  $y$  in both  $S$  and  $U$ , the resulting query  $q''$  is safe as well (Fig. 13c) with one single minimal plan  $P'' = \pi_{-x,y}^p \bowtie^p [S(x), \pi_{-z}^p \bowtie^p [R(x,y,z), U(x,z)]]$ . Note that  $P''$  has one projection less than  $P'$ . ■

Algorithm 3 applies Lemma 32 to dissociate eagerly on all functional dependencies, before it calls Algorithm 2.

**Theorem 34 (Algorithm 3)** *Algorithm 3 returns a sound and complete enumeration of minimal query plans in the presence of functional dependencies and deterministic relations.*

From the completeness of Algorithm 1, Algorithm 2, and Algorithm 3, and the fact that both Algorithm 2 and Algorithm 3 exploit equivalences in the partial dissociation order to reduce the number of minimal plans, it follows that exploiting schema knowledge when available can never increase the number of minimal plans.

**Algorithm:** FP (PlansWithFunctionalDependencies)  
**Input:** Query  $q(\mathbf{x}) : -R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$   
Set of FDs  $\Gamma$ , Schema information on deterministic tables  
**Output:** Set of all minimal query plans  $\mathcal{P}$

- 1 **forall**  $(\Gamma : \mathbf{y} \rightarrow \mathbf{z}) \in \Gamma$  and  $R_i \in \mathbf{R}$  **do**
- 2   **if**  $\mathbf{y} \subseteq \text{Var}(R_i)$  **then** dissociate  $R_i$  on all variables  $\mathbf{z} \setminus \mathbf{x}_i$
- 3   Let  $R'_i(\mathbf{x}'_i)$  be the resulting relations
- 4   Let  $q'(\mathbf{x}) : -R'_1(\mathbf{x}'_1), \dots, R'_m(\mathbf{x}'_m)$
- 5   DP( $q'$ )   // call to Algorithm 2

**Algorithm 3** uses knowledge of functional dependencies to apply selective dissociations before calling Algorithm 2.

### Corollary 35 (Minimal plans with schema knowledge)

*Knowledge about deterministic tables and functional dependencies cannot increase the number of minimal query plans.*

It is easy to see that Algorithm 3 returns one single query plan iff the query is safe, taking into account its structure, deterministic relations and functional dependencies. It is thus a *strict generalization of all known safe self-join-free conjunctive queries* [15, 51].

**Corollary 36 (Dichotomy)** *A query is safe if and only if Algorithm 3 returns one single plan.*

**Example 37** (Running example continued) Assume keys  $R(z,x)$  and  $S(u,y)$ , and hence the FDs  $z \rightarrow x$  and  $u \rightarrow y$  hold. Dissociating all dependent variables leads to a new query  $q_f$  (Fig. 12a) that has only two minimal safe dissociations (Fig. 12b and Fig. 12c) with two minimal plans (Fig. 12e) returned by Algorithm 3. If we know, in addition, that relations  $R$  and  $T$  are deterministic, then the query becomes  $q_{fd}$  which is safe (the hierarchy is shown in Fig. 12c) and has thus only one single plan (Fig. 12f). We encourage the reader to take a moment and study plus compare the incidence matrices Fig. 9b, Fig. 11b, and Fig. 12d carefully. ■

## 6 Multi-query Optimizations

So far, Algorithm 3 enumerates all minimal query plans. We then take the minimum score of those plans in order to calculate the propagation score  $\rho(q)$ . In this section, we develop three optimizations that can considerably reduce the necessary calculations needed to evaluate all minimal query

```

Recursive algorithm: SP (SinglePlan)
Input: Query  $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ 
Output: Single query plan  $P$ 
1 if  $m = 1$  then  $P \leftarrow \pi_{\mathbf{x}}^p R_1(\mathbf{x}_1)$  else
2   if  $q$  is disconnected then
3     Let  $q = q_1, \dots, q_k$  be the components connected by  $\text{EVar}(q)$ 
4     Let  $\text{HVar}(q_i) \leftarrow \text{HVar}(q) \cap \text{Var}(q_i)$ 
5      $P \leftarrow \bowtie^p [\text{SP}(q_1), \dots, \text{SP}(q_k)]$ 
6   else
7     Let  $\text{TopSets}(q) = \{\mathbf{y}_1, \dots, \mathbf{y}_j\}$ 
8     Let  $q'_i \leftarrow q_i$  with  $\text{HVar}(q'_i) \leftarrow \text{HVar}(q) \cup \mathbf{y}_i$ 
9     if  $j = 1$  then  $P \leftarrow \pi_{\mathbf{y}_1}^p \text{SP}(q'_1)$ 
10    else  $P \leftarrow \min [\pi_{\mathbf{y}_1}^p \text{SP}(q'_1), \dots, \pi_{\mathbf{y}_j}^p \text{SP}(q'_j)]$ 

```

**Algorithm 4** Optimization 1 recursively pushes the min operator into the leaves and generates one single query plan.

plans. Note that these three optimizations and the two optimizations from the previous section are orthogonal and can be arbitrarily combined in the obvious way.

### 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 top set  $\mathbf{y} \in \text{TopSets}(q)$  in line 10 of Algorithm 1, the adapted Algorithm 4 takes the minimum score over those top sets, for each tuple of the head variables in line 10. It thus creates one single query plan. Fig. 14 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. The adapted Algorithm 5 works as follows: it first traverses the whole single query plan (FindingCommonSubplans) and remembers each subplan by the subgoals used and its head variables in a HashSet HS (line 12). If it sees a subplan twice (line 12), it creates a new view for this subplan, mapping the subplan to a new view definition. The actual plan (ViewReusingPlan) then uses these views whenever possible (line 16). The order in which the views are created (line 5) assures that the algorithm also discovers and exploits *nested common subexpressions*. Fig. 15 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  $V_2$ .

```

Algorithm: UsingCommonSubplans
Input: Query  $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ 
Output: Ordered set of view definitions  $\mathcal{V}$ , final query plan  $P$ 
1  $\text{HS} \leftarrow \emptyset$  // HashSet of all subplans
2  $\text{HM} \leftarrow (\emptyset, \emptyset)$  // HashMap from subplans to unique view names
3  $\mathcal{V} \leftarrow \emptyset$  // Set of view definitions
4  $\text{FS}(q)$ 
5 foreach  $q_i \in \text{HM.keys}$  in increasing size of  $\text{HVar}(q_i)$  and  $\text{Var}(q_i)$  do
6    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\text{HM.val} = \text{ViewReusingPlan}(q_i)\}$ 
7  $P = \text{RP}(q)$ 

Recursive function: FS (FindingCommonSubplans)
Input: Query  $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ 
if  $q$  is disconnected then
9   Let  $q = q_1, \dots, q_k$  be the components connected by  $\text{EVar}(q)$ 
10  foreach  $q_i$  do  $\text{FS}(q_i(\mathbf{x}_i))$ 
11 else
12  if  $(m = 1 \wedge \mathbf{x} = \mathbf{x}_i) \vee \text{HM}(q) \neq \emptyset$  then return if  $q \in \text{HS} \wedge \text{HM}(q) = \emptyset$  then
13   $\text{HM}(q) \leftarrow$  new view name  $\text{HS} \leftarrow \text{HS} \cup \{q\}$ 
14  foreach  $\mathbf{y} \in \text{TopSets}(q)$  do
15  | Let  $q' \leftarrow q$  with  $\text{HVar}(q') \leftarrow \text{HVar}(q) \cup \mathbf{y}$ 
16  |  $\text{FS}(q')$ 

Recursive function: RP (ViewReusingPlan)
Input: Query  $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ 
Output: Query plan  $P$  that reuses views from HashMap HM
16 if  $\text{HM}(q) \neq \emptyset$  then  $P \leftarrow \text{HM}(q)$ 
17 else
18 | Insert here lines 1-11 from Algorithm 4, replacing SP with RP

```

**Algorithm 5** Optimizations 1 & 2 together create a query plan which re-uses several previously defined temporary views.

```

Algorithm: SR (SemiJoinReduction)
Input: Query  $q(\mathbf{x}) :- R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m)$ 
Output: Set of view definitions:  $BV, R_1^*, \dots, R_m^*$ ,
        New query  $q^*$  over those views
1  $BV = \bowtie [R_1(\mathbf{x}_1, p_1), \dots, R_m(\mathbf{x}_m, p_m)]$ 
2 foreach  $i \in [m]$  do
3 |  $R_i^* = \pi_{\mathbf{x}_i, p_i} BV(\text{Var}(q), \mathbf{p})$ 
4 Let  $q^*(\mathbf{x}) :- R_1^*(\mathbf{x}_1), \dots, R_m^*(\mathbf{x}_m)$ 
5  $\text{UsingCommonSubplans}(q^*)$  // applying Algorithm 5

```

**Algorithm 6** Optimization 3 performs a full deterministic semi-join reduction before the actual probabilistic query evaluation.

### 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. The idea of the third optimization is to first apply a *full semi-join reduction on the input relations* before starting the probabilistic plan evaluation from these *reduced input relations*. Algorithm 6 gives the pseudocode of this reduction. As it operates on the actual representation of the probabilistic tables, we explicitly write the probabilistic attributes of the input tables in our formalism in contrast to the rest of this paper:  $\pi_{\mathbf{x}_i, p_i}$  stands for the deterministic project with duplicate elimination on the variables of a relation  $R_i$  including its probabilistic attribute  $p_i$ , and  $\bowtie [\dots]$  for the natural join operator. The schema of the big view is thus:  $BV(\text{Var}(q), p_1, \dots, p_m)$ . Figure 16 shows this reduction applied to our running example.

$$\rho(q) = \min \left[ \begin{array}{l} \left[ \begin{array}{l} \pi_{-z}^p \bowtie^p [T(z), \min \left[ \begin{array}{l} \pi_{-x}^p \bowtie^p [R(x,z), \pi_{-u}^p \bowtie^p [U(u), \pi_{-y}^p \bowtie^p [S(y,u), M(x,y,z,u)]]] \\ \pi_{-u}^p \bowtie^p [U(u), \min \left[ \begin{array}{l} \pi_{-x}^p \bowtie^p [R(x,z), \pi_{-y}^p \bowtie^p [S(y,u), M(x,y,z,u)]]] \\ \pi_{-y}^p \bowtie^p [S(y,u), \pi_{-x}^p \bowtie^p [R(x,z), M(x,y,z,u)]]] \end{array} \right] \end{array} \right] \\ \pi_{-u}^p \bowtie^p [U(u), \min \left[ \begin{array}{l} \pi_{-y}^p \bowtie^p [S(y,u), \pi_{-z}^p \bowtie^p [T(z), \pi_{-x}^p \bowtie^p [R(x,z), M(x,y,z,u)]]] \\ \pi_{-x}^p \bowtie^p [R(x,z), \pi_{-y}^p \bowtie^p [S(y,u), M(x,y,z,u)]]] \end{array} \right] \end{array} \right] \end{array} \right]$$

**Fig. 14** Algorithm 4 (Optimization 1) evaluates only one single query plan by pushing the min operator down into the leaves.

$$\begin{array}{l} V_1(x,z,u) = \pi_{-y}^p \bowtie^p [S(y,u), M(x,y,z,u)] \\ V_2(y,z,u) = \pi_{-x}^p \bowtie^p [R(x,z), M(x,y,z,u)] \\ V_3(z,u) = \min \left[ \begin{array}{l} \pi_{-x}^p \bowtie^p [R(x,z), V_1(x,z,u)] \\ \pi_{-y}^p \bowtie^p [S(y,u), V_2(y,z,u)] \end{array} \right] \end{array} \quad \rho(q) = \min \left[ \begin{array}{l} \left[ \begin{array}{l} \pi_{-z}^p \bowtie^p [T(z), \min \left[ \begin{array}{l} \pi_{-x}^p \bowtie^p [R(x,z), \pi_{-u}^p \bowtie^p [U(u), V_1(x,z,u)]] \\ \pi_{-u}^p \bowtie^p [U(u), V_3(z,u)] \end{array} \right] \\ \pi_{-u}^p \bowtie^p [U(u), \min \left[ \begin{array}{l} \pi_{-y}^p \bowtie^p [S(y,u), \pi_{-z}^p \bowtie^p [T(z), V_2(y,z,u)]] \\ \pi_{-z}^p \bowtie^p [T(z), V_3(z,u)] \end{array} \right] \end{array} \right] \end{array} \right]$$

**Fig. 15** Algorithm 5 (Optimizations 1 & 2) exploits common subexpressions within a query plan by creating several possibly nested views.

$$\begin{array}{l} BV(x,y,z,u, p_R, p_S, p_T, p_U, p_M) :- R(x,z, p_R), S(y,u, p_S), T(z, p_T), \\ \quad U(u, p_U), M(x,y,z,u, p_M) \\ R^*(x,z, p_R) :- BV(x,y,z,u, p_R, p_S, p_T, p_U, p_M) \\ S^*(y,u, p_S) :- BV(x,y,z,u, p_R, p_S, p_T, p_U, p_M) \\ T^*(z, p_T) :- BV(x,y,z,u, p_R, p_S, p_T, p_U, p_M) \\ U^*(u, p_U) :- BV(x,y,z,u, p_R, p_S, p_T, p_U, p_M) \\ M^*(x,y,z,u, p_M) :- BV(x,y,z,u, p_R, p_S, p_T, p_U, p_M) \end{array}$$

**Fig. 16** Algorithm 6 (Optimization 3) performs a full semi-join reduction before the actual probabilistic evaluation. Note that these queries are deterministic and that the variables  $p_{R_i}$  stand here for the probabilistic attributes of tables, which we otherwise do not explicitly write.

## 7 Experiments

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

**Ranking quality.** We use *mean average precision* (MAP) to evaluate the quality of ranking of a method by comparing it against the ranking from exact probabilistic inference as ground truth (GT). MAP rewards rankings that place relevant items earlier; the best possible value is 1, and the worst possible 0 [46]. Average Precision at 10 (AP@10) is defined as  $AP@10 = \frac{\sum_{k=1}^{10} P@k}{n}$ , where  $P@k$  is the precision at  $k$ th answer returned. Averaging over several rankings yields MAP.

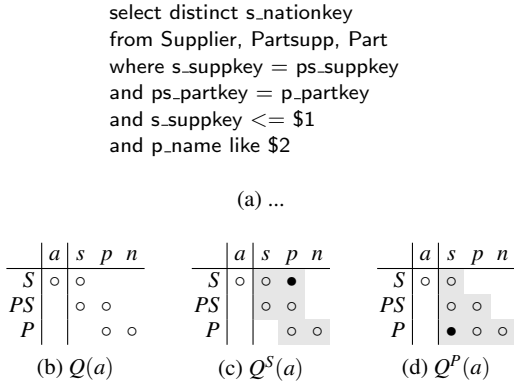
Scoring functions sometimes lead to ties between functions and, therefore, only partially ordered result lists. We use a variant of the analytic method proposed in [47] to calculate AP in the presence of ties, which is equivalent to calculating the mean AP over all allowed complete orders of the partial order imposed by the scores. We refer to their paper for details of the method. As baseline for no ranking, we assume all tuples to have the same score and thus be tied for the same position and call this *random average precision*.

**Exact probabilistic inference.** Whenever possible, we calculate GT rankings with a tool called SampleSearch [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 [30].

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

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

**Setup 1.** We use the TPC-H DBGEN data generator [3] to generate a 1GB database to which we add a column  $P$  for each table and store it in PostgreSQL 9.2 [2]. We assign to each input tuple  $i$  a random probability  $p_i$ , uniformly chosen from the interval  $[0, p_{i\max}]$  resulting in an expected average input probability  $\text{avg}[p_i] = p_{i\max}/2$ . By using databases with  $\text{avg}[p_i] < 0.5$ , we can avoid answer probabilities close to 1 for queries with very large lineages. We use the following



**Fig. 17** Parameterized Deterministic SQL query  $Q(a)$  over TPC-H. Incidence matrices for TPC-H query  $Q(a)$  and its two minimal safe dissociations from either dissociating table  $S$  or table  $P$ .

parameterized query (Fig. 17a):

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

Relations  $S$ ,  $PS$  and  $P$  represent tables Supplier, PartSupp and Part, respectively. Variable  $a$  stands for attribute nationkey (“answer tuple”),  $s$  for suppkey,  $u$  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 suppkey  $\leq \$1$  that supply parts with a name like  $\$2$ ?” Parameters  $\$1$  and  $\$2$  allow us to change the lineage sizes: 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  $\text{MAP@10} = 0.220$ .

This query has the following two minimal query plans (Fig. 17):

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

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

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 the our three optimizations against evaluation of all plans for  $k$ -chain queries and  $k$ -star queries over varying database sizes (data complexities) and varying query sizes (query complexities). The  $k$ -chain queries have arity = 2 and several results, whereas

the star queries have arity = 1 and cardinality = 1, representing a Boolean query:

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

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

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 are uniformly randomly drawn from the range  $\{0, 1, \dots, N-1\}$ . This parameter determines the *selectivity* and is varied as to keep the *answer cardinality constant* around 20-50 for chain queries, and the answer probability between 0.90 and 0.95 for star queries. For the data complexity experiments, we vary the number of tuples  $n$  per table from 100 to  $10^6$ . For the query complexity experiments, we vary  $k$  from 2 and 7, or 2 and 8 for chain and star queries, respectively. For these experiments, the optimized (and often *extremely long*) SQL statements are “calculated” in JAVA and then sent to Microsoft SQL server [1] (we have used Microsoft SQL server for these experiments due to its superior handling of complex queries).

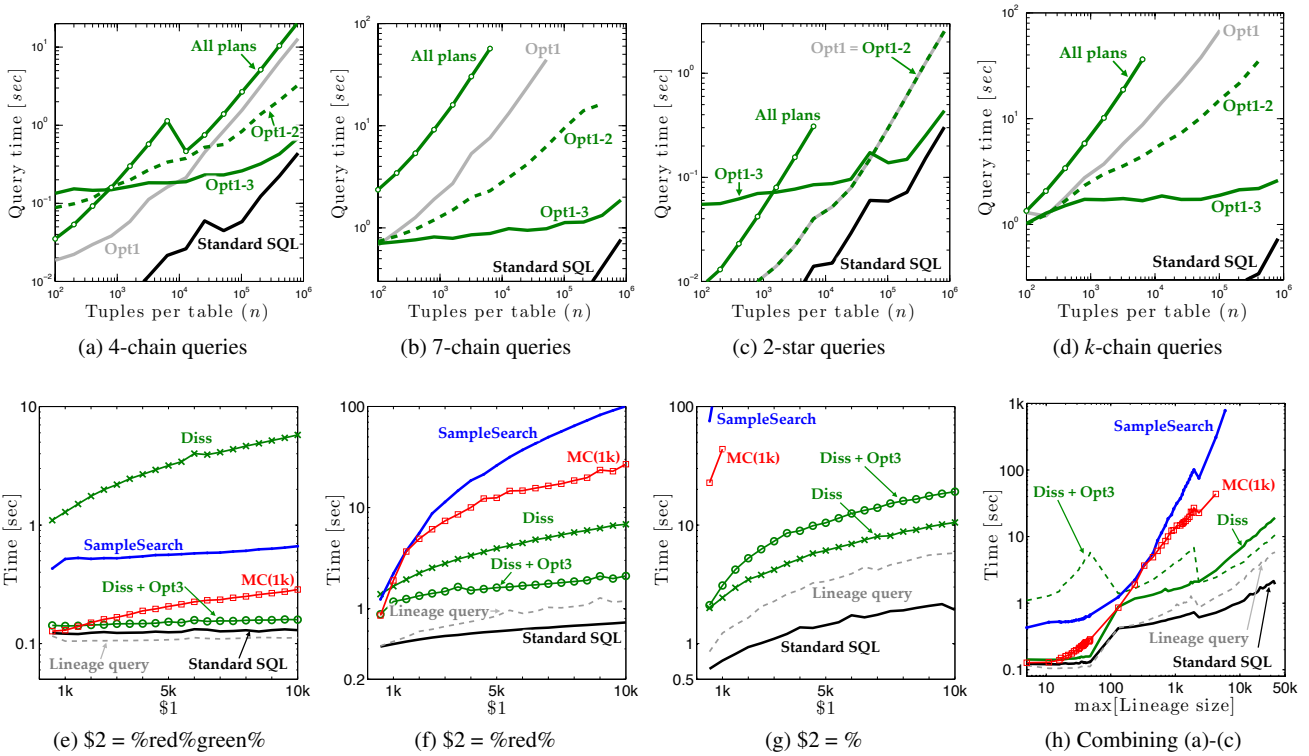
## 7.1 Runtime experiments

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

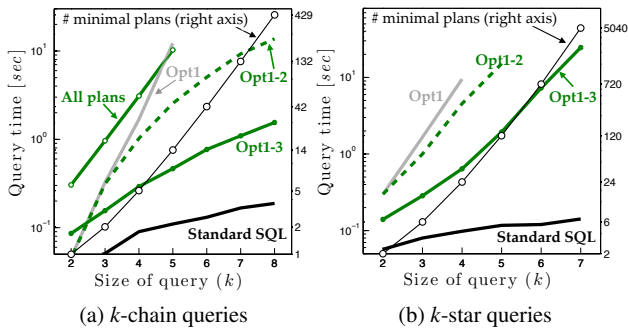
*Result 1.* Combining plans (Opt. 1) and using intermediate views (Opt. 2) almost always speeds up query times. The semi-join reduction (Opt. 3) slows down queries with high selectivities, but considerably speeds up queries with small selectivities, bringing probabilistic query evaluation close to deterministic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

Fig. 18a to through Fig. 18d show the results on setup 2 for increasing database sizes or query sizes. Figure 19 shows results on setup 2 for increasing query sizes. For example, Fig. 18b shows the performance of computing a 7-chain query which has 132 safe dissociations. Evaluating each of these queries separately takes a long time, while our optimization techniques brings evaluation time close to deterministic query evaluation. Especially on larger databases, where the running time is I/O bound, the penalty of the probabilistic inference is only a factor of 2-3 in this example. Notice here the trade-off between optimization 1,2 and optimization 1,2,3: Optimization 3 applies a full semi-join reduction on the input relations before starting the probabilistic plan evaluation from these reduced input relations. This operation imposes a rather large constant overhead, both at the query optimizer and at query execution. For larger databases (but constant selectivity), this overhead is amortized. 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





**Fig. 18** 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.



**Fig. 19** 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 minimal plans (!) for a 7-star query).

could not produce a query plan”). In practice, this suggests that dissociation allows a large space of optimizations depending on the query and particular database instance that can conservatively extend the space of optimizations performed today in deterministic query optimizers.

Fig. 18e to Fig. 18g 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 rank-

ing (“Standard SQL”) on setup 1. As experimental platform, we use 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  $\$2 \in \{\text{'%red%green%'}, \text{'%red%'}, \text{'%'}\}$  and varied  $\$1 \in \{500, 1000, \dots, 10k\}$ . Fig. 18h 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. 18e 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. 18f shows overhead of up to 2).

We would like to draw here an important connection to [51] who introduces the idea of “lazy plans” and show 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 (Opt. 3) *serve the same purpose* with similar performance improvements and also for safe queries. The advantage of our semi-join reductions, however, is that we do not require any modifications to the query engine.

	\$2	%red%green%		%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
<b>SampleSearch</b> [sec]		0.43	0.66	1.23	100.71	75.47	–
<b>MC(1k)</b> [sec]		0.13	0.29	0.86	26.87	22.75	–
<b>Dissociation &amp; SJ</b> [sec]		0.14	0.16	0.88	2.11	2.11	19.14
<b>Dissociation</b> [sec]		1.10	5.76	1.39	6.83	2.00	10.52
<b>Lineage SQL</b> [sec]		0.12	0.11	0.43	1.19	0.86	5.80
<b>Deterministic SQL</b> [sec]		0.12	0.13	0.42	0.73	0.61	1.93

Fig. 20 Overview timing results TPC-H.

*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.*

Fig. 18e to Fig. 18h show that SampleSearch does not scale to larger lineages as the performance of exact probabilistic inference depends on the tree-width of the Boolean lineage formula, which generally increases with the size of the data. In contrast, dissociation is *independent of the treewidth*. For example, SampleSearch needed 780 sec for calculating the ground truth for a query with  $\max[\text{lin}] = 5.9\text{k}$  for which dissociation took 3.0 sec, and MC(1k) took 42 sec for a query with  $\max[\text{lin}] = 4.2\text{k}$  for which dissociation took 2.4 sec. Dissociation takes only 10.5 sec for our largest query  $\$2 = \text{'%'} and  $\$1 = 10\text{k}$  with  $\max[\text{lin}] = 35\text{k}$ . Retrieving the lineage for that query alone takes 5.8 sec, which implies that any probabilistic method that evaluates the probabilities outside of the database engine needs to issue this query to retrieve the DNF for each answer and would thus have to evaluate lineages of sizes around 35k in only 4.7 (= 10.5 - 5.8) sec to be faster than dissociation.<sup>10</sup>$

*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 3 sec with common views instead of our reported 0.88 sec for materialized views. We also found that using standard database-provided aggregates (which requires use to use the logarithm for products) instead of user-defined aggregates considerably 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 SQL 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

<sup>10</sup> 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.

of the improvements, the query  $\$1 = 10\text{k}$  and  $\$2 = \text{'%'}$  would take 42.2 sec instead of 20.7 with semi-join reduction, and 32.5 sec 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 *without any modifications to the database engine nor performing any calculations outside the database*.

## 7.2 Ranking experiments

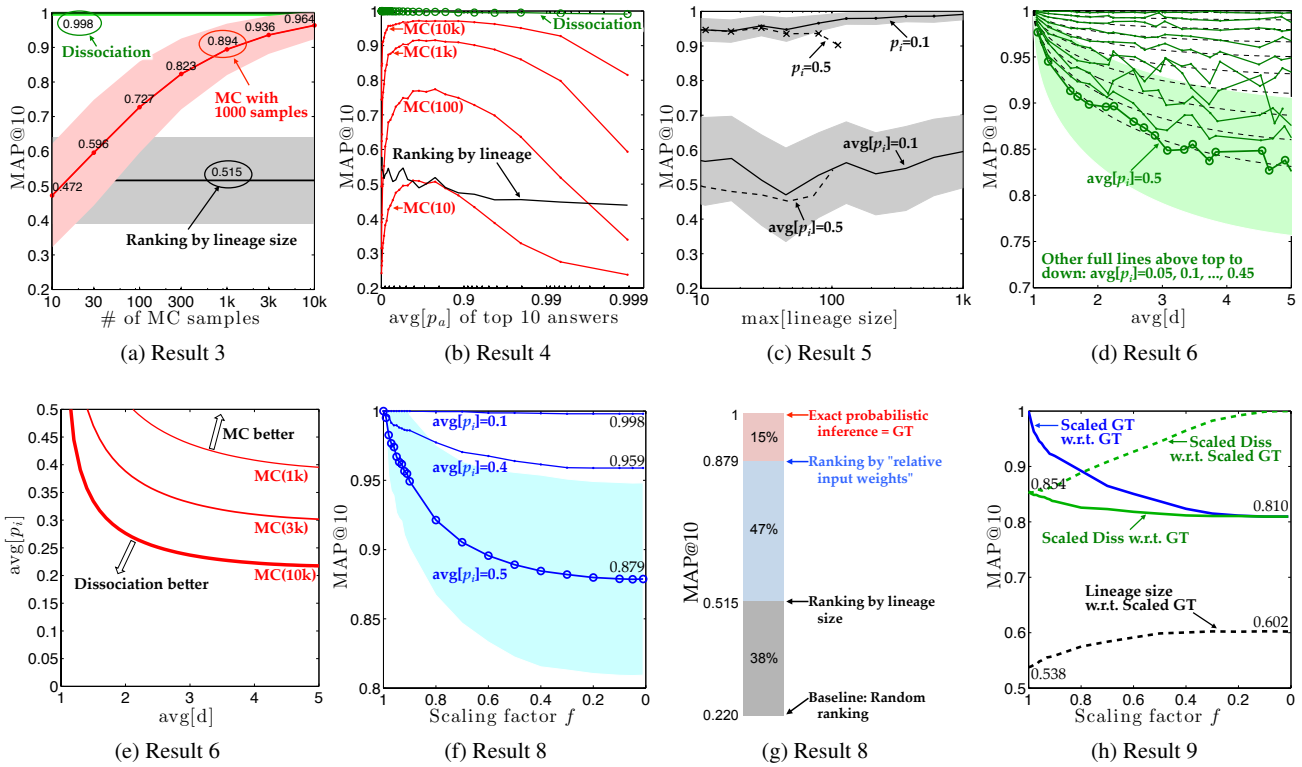
For the following experiments, we are limited to those query parameters  $\$1$  and  $\$2$  for which we can get the ground truth (and results from MC) in acceptable time. We systematically vary  $p_{i\max}$  between 0.1 and 1 (and thus  $\text{avg}[p_i]$  between 0.05 and 0.5) and evaluate the rankings 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.*

Fig. 21a shows averaged results of our probabilistic methods for  $\$2 = \text{'%red%green%'}$ .<sup>11</sup> Shaded areas indicate standard deviations and the x-axis shows varying numbers of MC samples. We only used those data points for which  $\text{avg}[p_a]$  of the top 10 ranked tuples is between 0.1 and 0.9 according to ground truth ( $\approx 6\text{k}$  data points for dissociation and lineage,  $\approx 60\text{k}$  data points for MC, as we repeated each MC simulation 10 times) as this the best regime for MC according to Result 4. Figure 22 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

<sup>11</sup> Results for MC and other parameters of  $\$2$  are similar. However, the evaluation time for the experiments becomes quickly infeasible.



**Fig. 21** Timing results: (a)-(c) For increasing database sizes and constant cardinalities, our optimizations approach deterministic SQL performance. (d) Our optimizations can even evaluate very large number of minimal plans efficiently (here shown up to 429 for a 8-chain query). (e)-(h) For the TPC-H query, the best evaluation for dissociation is within a factor of 6 of that for deterministic query evaluation. (i)-(p) Ranking experiments on TPC-H: Assumptions for from each plot and conclusions are described below each respective result in the text.

strings, such as '%r%g%r%a%n%d%' and '%re%re%'. The average MAP over all 28 choices for parameters \$2 is 0.997 for ranking by dissociation and 0.520 for ranking by lineage size ( $\approx 100k$  data points). Most of those queries have too large of a lineage to evaluate MC. Note that ranking by lineage always returns the same ranking for given parameters \$1 and \$2, but the GT ranking would change with different input probabilities.

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

Fig. 21b shows the AP as a function of  $\text{avg}[p_a]$  of the top 10 ranked tuples according to ground truth by logarithmic scaling of the x-axis (each point in the plot averages AP over  $\approx 450$  experiments for dissociation and lineage and over  $\approx 4.5k$  experiments for MC). We see that MC performs increasingly poor for ranking answer tuples with probabilities close to 0 or 1 and even approach the quality of random ranking ( $\text{MAP}@10 = 0.22$ ). This is so because 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. 21a.

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

Fig. 21c 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. Fig. 21d 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 1000s). 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 area for  $\text{avg}[p_i] = 0.5$ . We see that the quality is very dependent on  $\text{avg}[p_i]$ , as predicted by Prop. 23.

Notice that one should not confuse the AP score of each of the two plans taken separately, with the AP score of the min between the two plans; the ranking produced by the minimal probability of the two plans can be much better than each of the two rankings produced by each plan separately. 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. 21g) to only 0.995, which shows the value of taking the minimum score for each tuple *individually*.

*Question 4* When does dissociation perform better than MC?

*Result 7. Dissociation performs very well and notably better than MC(10k) if either  $\text{avg}[d]$  or  $\text{avg}[p_i]$  is small.*

Fig. 21e 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

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
	$\approx 0.99$	10k	0.945	0.046
	$\approx 0.99$	3k	0.897	0.059
$\approx 0.99$	1k	0.827	0.076	
Lineage size	$p_i$		MAP@10	stdv
	random all equal		0.520 0.949	0.130 0.033
Random ranking			MAP@10	stdv
			0.220	0.112

Fig. 22 Quality results TPC-H: Our three methods, their respectively most important parameters, and their average ranking qualities.

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. 21a, i.e. assuming  $0.1 < \text{avg}[p_a] < 0.9$  for MC (recall that MC would perform notably worse if including data points with  $\text{avg}[p_a] > 0.9$ ). 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. 21c). Thus, for large lineages, we need small input probabilities to have meaningful interpretations. And for small input probabilities, dissociation considerably outperforms any other method.

*Question 5* How much would the ranking change according to exact probabilistic inference if we scale down all input tuples?

*Result 8. 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 on average  $\text{avg}[d] \approx 3$  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  $1 > f > 0$ , then compare the new ranking against GT. Fig. 21f shows that if all input probabilities are already small (and dissociation already works well), then scaling has little to no 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 they lose their relative importance 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 ( $\text{MAP} = 0.529$ ) and the ranking on a scaled database for  $f \rightarrow 0$  ( $\text{MAP} = 0.879$ ) 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  $\text{MAP} = 0.220$  as baseline for random ranking, 38% of the ranking quality can be found by the lineage size alone vs. 85% by the lineage size plus the relative weights of input tuples. The remaining 15% come from the actual probabilities (Fig. 21g). While these particular numbers only hold for this particular scenario (the average of the chosen 7 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 6* 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 9. 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. 21h, 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 Prop. 23). 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 (thus 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.* Current approaches to query evaluation on probabilistic databases can be classified into three categories (Fig. 23 ): (i) *incomplete approaches* identify tractable cases either at the query-level [15, 16, 26] or the

	all queries	all data	exact score	Performance	rel. algebra	
Safe query plans [15, 16]	•	•	•	•	•	} (1) incomplete
Read-once formulas [50, 66]	•		•	•		
Exact prob. inference [42, 51]	•	•	•			} (2) slow
Approx. prob. inference [52]	•	•	○			
Monte Carlo [41, 45, 59]	•	•	○			} (3) approximate

**Fig. 23** 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.

data-level [50, 62, 66]; (ii) *exact approaches* [5, 42, 50, 51, 65] work well on queries with simple lineage expressions, but perform poorly on database instances with complex lineage expressions. (iii) *approximate approaches* either apply general purpose sampling methods [41, 44, 45, 59], or approximate the number of models of the Boolean lineage expression [25, 52, 60]. We note that the approach described in this paper generalizes several of these techniques: our algorithm returns the exact score if the query is safe [15, 51] or data-safe [42].

*Lifted inference.* Lifted inference was introduced in the AI literature as an approach to efficient probabilistic inference that exploits symmetries at the grounded level of a first-order formula [55]. 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 [20] correspond to safe plans. See [19] for a recent discussion on the similarities and differences.

*Representing Correlations.* The most popular approach to represent correlations between the tuples in a probabilistic database is by a Markov Logic network (MLN), which is a set of *soft constraints* [22]. Quite remarkably, all complex correlations introduced by an MLN can be rewritten into a query over a tuple-independent probabilistic database [33, 38, 43]. Using these techniques, the work described in this paper extends to MLNs as well.

*Dissociation.* We introduced dissociation first in the workshop paper [29] as a generalization of graph propagation [21] to hypergraphs. In [30] we provide 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 as upper bounds, 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* [18]. See [30] for a detailed discussion on the similarities and differences. Query dissociation is also related to recent work that gives upper bounds on the partition function of a Markov random field. [70] develops a method to *obtain optimal upper bounds* by replacing the original distribution using a *collection of tractable distributions*, i.e. such for which the partition function can be calculated efficiently by a recursive algorithm. In our work, efficient approximations of distributions at the schema level are those that allow a *safe query plan*, and can thus be evaluated in a standard DBMS.

## 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 propagation 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-0915054 and IIS-1115188. We like to thank Abhay Jha for help with the experiments in the workshop version of this paper [29]. WG would also like to thank Manfred Hauswirth for a small side comment in 2007 that was crucial for the development of the ideas in this paper.

## References

1. Microsoft SQL Server 2008 R2: <http://www.microsoft.com/sqlserver/>.
2. PostgreSQL 9.2: <http://www.postgresql.org/download/>.
3. TPC-H benchmark: <http://www.tpc.org/tpch/>.
4. Antoine Amarilli, Yael Amerdamer, and Tova Milo. Uncertainty in crowd data sourcing under structural constraints. In *DASFAA Workshops*, pages 351–359, 2014.
5. Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.
6. Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, pages 1479–1480, 2007.
7. Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Model counting of query expressions: Limitations of propositional methods. In *ICDT*, pages 177–188, 2014.
8. Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.
9. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
10. Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.
11. Yang Chen and Daisy Zhe Wang. Knowledge expansion over probabilistic knowledge bases. In *SIGMOD*, pages 649–660, 2014.
12. Charles J. Colbourn. *The combinatorics of network reliability*. Oxford University Press, New York, 1987.
13. Yves Crama and Peter L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, 2011.
14. Fabio Crestani. Application of spreading activation techniques in information retrieval. *Artif. Intell. Rev.*, 11(6):453–482, 1997.
15. Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.
16. Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30, 2012.
17. DeepDive: <http://deepdive.stanford.edu/>.
18. Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *UAI*, pages 131–141, 2012.
19. Guy Van den Broeck and Dan Suciu. Lifted probabilistic inference in relational models. In *UAI tutorials*, 2014.
20. Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*, pages 2178–2185, 2011.
21. Landon Detwiler, Wolfgang Gatterbauer, Brent Louie, Dan Suciu, and Peter Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In *ICDE*, pages 1235–1238, 2009.
22. Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool Publishers, 2009.
23. X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmman, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.
24. Beyza Ermis and Guillaume Bouchard. Scalable binary tensor factorization. In *UAI*, 2014.
25. Robert Fink and Dan Olteanu. On the optimal approximation of queries using tractable propositional languages. In *ICDT*, pages 174–185, 2011.
26. Robert Fink and Dan Olteanu. A dichotomy for non-repeating queries with negation in probabilistic databases. In *PODS*, pages 144–155, 2014.

27. Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.
28. Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and Turbo Belief Propagation, June 2014. (CoRR abs/1406.7288).
29. Wolfgang Gatterbauer, Abhay K. Jha, and Dan Suciu. Dissociation and propagation for efficient query evaluation over probabilistic databases. In *Proc. 4th International VLDB workshop on Management of Uncertain Data (MUD)*, pages 83–97, 2010.
30. Wolfgang Gatterbauer and Dan Suciu. Oblivious bounds on the probability of Boolean functions. *ACM Trans. Database Syst. (TODS)*, 39(1):5, 2014.
31. V. Gogate and R. Dechter. SampleSearch: Importance sampling in presence of determinism. *Artificial Intelligence*, 175(2):694–729, 2011.
32. Vibhav Gogate and Pedro Domingos. Formula-based probabilistic inference. In *UAI*, pages 210–219, 2010.
33. Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. In *UAI*, pages 256–265, 2011.
34. Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of Satisfiability*, pages 633–654, 2009.
35. Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250, 2010.
36. Erich Grädel, Yuri Gurevich, and Colin Hirsch. The complexity of query reliability. In *PODS*, pages 227 – 234, 1998.
37. Ramanathan V. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *WWW*, pages 403 – 412, 2004.
38. Adnan Darwiche Guy Van den Broeck, Wannes Meert. Skolemization for weighted first-order model counting. In *KR*, 2014. (to appear).
39. Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.
40. Manfred Jaeger and Guy Van den Broeck. Liftability of probabilistic inference: Upper and lower bounds. In *StarAI*, 2012.
41. Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.
42. Abhay Jha, Dan Olteanu, and Dan Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, pages 323–334, 2010.
43. Abhay Jha and Dan Suciu. Probabilistic databases with markovviews. *PVLDB*, 5(11):1160–1171, 2012.
44. Shantanu Joshi and Christopher M. Jermaine. Sampling-based estimators for subset-based queries. *VLDB J.*, 18(1):181–202, 2009.
45. Oliver Kennedy and Christoph Koch. Pip: A database system for great and small expectations. In *ICDE*, pages 157–168, 2010.
46. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
47. Frank McSherry and Marc Najork. Computing information retrieval performance measures efficiently in the presence of tied scores. In *ECIR*, pages 414–421, 2008.
48. Guido Moerkotte. Building query compilers. Draft version 03.03.09, Sept. 2009.
49. Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.
50. Dan Olteanu and Jiewen Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.
51. Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.
52. Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.
53. Jeff Pasternack and Dan Roth. Knowing what to believe (when you already know something). In *COLING*, pages 877–885, 2010.
54. Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, San Mateo, Calif., 1988.
55. David Poole. First-order probabilistic inference. In *IJCAI*, pages 985–991, 2003.
56. M. R. Quillian. Semantic memory. In *Semantic Information Processing*, pages 227–270. MIT Press, 1968.
57. Rohit Raghunathan, Sushovan De, and Subbarao Kambhampati. Bayesian networks for supporting query processing over incomplete autonomous databases. *J. Intell. Inf. Syst.*, 42(3):595–618, 2014.
58. Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Query evaluation on probabilistic databases. *IEEE Data Eng. Bull.*, 29(1):25–31, 2006.
59. Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.
60. Christopher Ré and Dan Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.
61. Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.
62. Sudeepa Roy, Vittorio Perduca, and Val Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.
63. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, chapter Learning internal representations by error propagation, pages 318–362. MIT Press, 1986.
64. Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23 – 34, 1979.
65. Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.
66. Prithviraj Sen, Amol Deshpande, and Lise Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.
67. Ajit Paul Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.
68. Julia Stoyanovich, Susan B. Davidson, Tova Milo, and Val Tannen. Deriving probabilistic databases with inference ensembles. In *ICDE*, pages 303–314, 2011.
69. Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.
70. Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.
71. Jason Weston, Andre Elisseeff, Dengyong Zhou, Christina S Leslie, and William Stafford Noble. Protein ranking: from local to global structure in the protein similarity network. *Proc Natl Acad Sci U S A*, 101(17):6559–63, Apr 2004.
72. Xiaoxin Yin, Jiawei Han, and Philip S. Yu. Truth discovery with multiple conflicting information providers on the web. *IEEE Trans. Knowl. Data Eng.*, 20(6):796–808, 2008.
73. Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, pages 277–288, 2014.
74. Ce Zhang and Christopher Ré. Towards high-throughput Gibbs sampling at scale: a study across storage managers. In *SIGMOD*, pages 397–408, 2013.

## A Nomenclature

$R, S, T, U$	relational tables
$r_i, s_i, t_i, u_i$	tuple identifiers
$A, B, C$	attribute names
$a, \dots, f, s, t$	constants
$s, t$	source and target nodes
$x, y, z$	variables
$q$	query
$g$	atom or subgoal
$sg(x)$	set of subgoals that contain $x$
$\text{Var}(q)$	set of variables of a query $q$ or subgoal $g$
$\text{HVar}(q)$	set of head variables of a query $q$ or a plan $P$
$\text{EVar}(q)$	set of existential variables: $\text{EVar}(q) = \text{Var}(q) - \text{HVar}(q)$
$\text{TopSets}(q)$	set of top sets of query $q$ : a top set is a minimal subset of $\text{EVar}(q)$ , removing of which disconnects $q$
$\text{SVar}(q)$	separator variables of a query $q$ : only those existential variables that appear in <i>all</i> probabilistic subgoals
$p$	concrete probability
$r(q)$	reliability score of $q$
$\rho(q)$	propagation score of $q$
$\phi, \psi$	Boolean expression
$\mathbb{P}[\phi]$	probability of a Boolean expression
$m$	number of subgoals
$n$	number of variables
$D$	database instance
$A$	active domain
$\Delta$	dissociation, collection of $m$ sets of variables, $\Delta = (y_1, \dots, y_m)$ with $y_i \subseteq \text{Var}(q) - \text{Var}(g_i)$
$q^A, q^{(i)}$	dissociated query
$P$	query plan
$\bowtie^P[\dots]$	probabilistic join operator in prefix notation
$\pi_x^p, \pi_y^p$	probabilistic project operators: onto $\mathbf{x}$ , or project $\mathbf{y}$ away
$\text{score}(P)$	score of a query plan
$\text{JVar}$	join variables for a join operator
$\text{RVar}(P)$	root variables of plan $P$ : $P = \pi_x^p P'$
$\mathcal{P}$	set of plans
$q_P$	query consisting of all atoms mentioned in plan $P$
$R_i^{y_i}$	dissociated relation $R_i(\mathbf{x}_i)$ on variables $\mathbf{y}_i$ : $R_i(\mathbf{x}_i, \mathbf{y}_i)$
$\tilde{R}, \tilde{x}$	short notation for dissociated relation or variable
$\mathbf{x}$	unordered set or ordered tuple
$[a/x]$	substitute value $a$ for variable $x$

## B Section 2: Proof Proposition 5

*Proof (Prop. 5: Safety)* (1) is proven in [15]; we prove here only (2):

(a) Hierarchical query  $\Rightarrow$  unique safe plan: We prove the following statement by induction: Let  $\mathbf{x}$  be a set of *root variables* for a query  $q(\mathbf{x})$ , i.e. every variable in  $\mathbf{x}$  occurs in every atom in  $q$ ; then  $q(\mathbf{x})$  admits a unique safe plan either as  $\bowtie^P[P_1, \dots, P_k]$  or as  $\pi_x^p P$ : Define a graph where the nodes are the atoms of  $q$  and any two nodes are connected by an edge iff they share an existential variable, i.e. a variable not occurring in  $\mathbf{x}$ . If the graph has  $k$  connected components represented by the queries  $q_1, \dots, q_k$ , then  $q \equiv \bowtie^P[q_1, \dots, q_k]$ , and we apply induction hypothesis to each  $q_i(\mathbf{x})$ . If the graph has a single connected component with additional variables  $\mathbf{y} \neq \emptyset$ , then  $q \equiv \pi_x^p q(\mathbf{x}, \mathbf{y})$ , and we apply induction hypothesis to  $q(\mathbf{x}, \mathbf{y})$ . Finally, if the graph has a single component and only the variables  $\mathbf{x}$ , then  $q$  has a single atom, hence  $q \equiv R(\mathbf{x})$ .

(b) Safe plan  $\Rightarrow$  hierarchical query: We construct inductively  $q$  from its derivation by noting that  $\bowtie^P[P_1, \dots, P_k] \equiv q_1 \wedge \dots \wedge q_k$ , where  $q_1, \dots, q_k$  are the hierarchical queries obtained inductively from

$P_1, \dots, P_k$ , and  $\pi_x^p P \equiv \exists \mathbf{x}. q$ , where  $q$  is obtained inductively from  $P$ . It is easy to check inductively that all resulting queries are hierarchical.  $\square$

## C Section 3: Proof Theorem 12

Before we prove Theorem 12, we need to develop some notions and lemmas for Boolean functions and their probabilistic interpretation.

### C.1 Preliminaries

*Boolean notions* [13]. Let  $\mathbf{x} = \{x_1, \dots, x_n\}$  be a set of  $n$  Boolean variables. We use the bar sign (e.g.  $\bar{\mathbf{x}}$ ) to denote an ordered or unordered set, depending on the context. A *truth assignment* or valuation  $\theta$  for  $\mathbf{x}$  is an  $n$ -vector in  $\{0, 1\}^n$ , i.e. it is a function  $\theta : \mathbf{x} \rightarrow \{0, 1\}^n$  where  $\theta_i = \theta(x_i)$  denotes the value assigned to  $x_i$  by  $\theta$ . A *positive term* or conjunct is  $c = \bigwedge_{i \in \mathbf{c}} x_i$ , where  $\mathbf{c} \subseteq \mathbf{x}$ . Note that variable  $c$  represents a conjunct, whereas the set  $\mathbf{c}$  represents the variables of  $c$ . A *positive DNF* (Disjunctive Normal Form) of size  $m$  is an expression of the form  $\bigvee_{j=1}^m c_j = \bigvee_{j=1}^m \left( \bigwedge_{i \in \mathbf{c}_j} x_i \right)$ , where each  $c_j$  ( $j \in \{1, \dots, m\}$ ) is a positive term of the DNF. A *positive  $k$ -uniform DNF* is one where each term contains exactly  $k$  variables. A *positive  $k$ -partite  $k$ -uniform DNF* is one where the set of variables  $\mathbf{x}$  can be partitioned into  $k$  sets  $(\mathbf{x}_1, \dots, \mathbf{x}_k)$  so that each term consists of exactly  $k$  variables with each variable coming from a different partition.

*Event expressions.* We assign to each Boolean variable  $x_i$  a *primitive event* (we do not formally distinguish between the independent random variable  $x_i$  and the event  $x_i$  that it is true) which is true with probability  $\mathbb{P}[x_i] = p_i$ . All primitive events are assumed to be independent, i.e.  $\mathbb{P}[x_i \wedge x_j] = \mathbb{P}[x_i] \cdot \mathbb{P}[x_j] = p_i \cdot p_j, \forall i, j \in \{1, \dots, n\}$  with  $i \neq j$ . We are interested in the computation of probabilities of *composed events* [27] and write  $\phi(\mathbf{x})$  to indicate that  $\mathbf{x} = \text{Var}(\phi)$  is the set of variables appearing in the expression  $\phi$ . Our focus is on calculating the probability of positive DNF event expressions. Given independence of primitive events, the probability of a *conjunct*  $c$  is

$$\mathbb{P}[c] = \prod_{i \in \mathbf{c}} \mathbb{P}[x_i] = \prod_{i \in \mathbf{c}} p_i$$

Using the inclusion-exclusion principle, the event probability for a *positive DNF* is then

$$\mathbb{P}\left[\bigvee_{j=1}^m c_j\right] = \sum_{k=1}^m (-1)^{k-1} \sum_{1 \leq j_1 < \dots < j_k \leq m} \mathbb{P}\left[\bigwedge_{i=1}^k c_{j_i}\right] \quad (3)$$

*Correlations and positive DNF dissociation.* Two events  $\phi_1$  and  $\phi_2$  are *positively correlated* iff

$$\mathbb{P}[\phi_1 \wedge \phi_2] > \mathbb{P}[\phi_1] \cdot \mathbb{P}[\phi_2],$$

By application of the inclusion-exclusion principle, it follows

$$\mathbb{P}[\phi_1 \vee \phi_2] < 1 - \mathbb{P}[\neg \phi_1] \cdot \mathbb{P}[\neg \phi_2]$$

**Lemma 38 (Positive correlations of terms)** *Two positive terms  $c_1$  and  $c_2$  over the random variables  $\mathbf{x}$  are positively correlated if and only if neither of them has probability 0, and they have at least one variable  $x_i$  in common with  $p_i \neq 1$ .*



*Proof (Lemma 38)* Assume  $c_1$  and  $c_2$  have a possibly empty set  $\mathbf{c}_{1 \cap 2} := \mathbf{c}_1 \cap \mathbf{c}_2$  of variables in common. We use  $\mathbf{c}_{1 \setminus 2}$  for  $\mathbf{c}_1 \setminus \mathbf{c}_2$  and can write  $\mathbb{P}[c_1] = \prod_{i \in \mathbf{c}_1} p_i = \prod_{i \in \mathbf{c}_{1 \setminus 2}} p_i \cdot \prod_{i \in \mathbf{c}_{1 \cap 2}} p_i$ . Hence,

$$\mathbb{P}[c_1] \cdot \mathbb{P}[c_2] = \prod_{i \in \mathbf{c}_{1 \setminus 2}} p_i \cdot \prod_{i \in \mathbf{c}_{2 \setminus 1}} p_i \cdot \prod_{i \in \mathbf{c}_{1 \cap 2}} p_i^2$$

whereas

$$\mathbb{P}[c_1 \wedge c_2] = \prod_{i \in \mathbf{c}_{1 \setminus 2}} p_i \cdot \prod_{i \in \mathbf{c}_{2 \setminus 1}} p_i \cdot \prod_{i \in \mathbf{c}_{1 \cap 2}} p_i$$

Therefore,

$$\mathbb{P}[c_1] \cdot \mathbb{P}[c_2] = \mathbb{P}[c_1 \wedge c_2] \cdot \prod_{i \in \mathbf{c}_{1 \cap 2}} p_i$$

from which the proposition follows.  $\square$

**Corollary 39 (Positive correlations of terms)** For any two positive terms  $c_1$  and  $c_2$  over the random variables  $\mathbf{x}$ , the following hold:

$$\mathbb{P}[c_1 \wedge c_2] \geq \mathbb{P}[c_1] \cdot \mathbb{P}[c_2]$$

$$\mathbb{P}[c_1 \vee c_2] \leq 1 - \mathbb{P}[\neg c_1] \cdot \mathbb{P}[\neg c_2]$$

**Definition 40 (Expression dissociation)** Assume a Boolean expression  $\phi$  over variables  $\mathbf{x}$ . A dissociation of  $\phi$  is a new expression  $\phi'$  over variables  $\mathbf{x}'$  so that there exists a substitution  $\theta : \mathbf{x}' \rightarrow \mathbf{x}$  that transforms the new into the original expression:  $\phi'(\theta(\mathbf{x}')) = \phi(\mathbf{x})$ . The probability of the new event expression  $\mathbb{P}[\phi']$  is evaluated by assigning each new variable independently the probability corresponding to its substitution in its event expression:  $\mathbb{P}[x'] = \mathbb{P}[\theta(x')]$ ,  $\forall x' \in \mathbf{x}'$ .

*Example 41 (Expression dissociation)* Take the two DNF expressions:

$$\phi(x_1, x_2, x_3, x_4) = x_1 x_3 \vee x_1 x_4 \vee x_2 x_4$$

$$\phi'(x_1, x_2, x_3, x_4, x'_4) = x_1 x_3 \vee x_1 x_4 \vee x_2 x'_4$$

Then  $\phi'(\mathbf{x}')$  is a dissociation of  $\phi(\mathbf{x})$  because  $\phi(\mathbf{x}) = \phi'(\theta(\mathbf{x}'))$  for the substitution  $\theta = \{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4), (x'_4, x_4)\}$ . Further,  $\mathbb{P}[\phi] = p_1 p_3 + p_1 p_4 + p_2 p_4 - p_1 p_3 p_4 - p_1 p_3 p_2 p_4 - p_1 p_4 p_2 + p_1 p_3 p_4 p_2$ , whereas  $\mathbb{P}[\phi'] = p_1 p_3 + p_1 p_4 + p_2 p_4 - p_1 p_3 p_4 - p_1 p_3 p_2 p_4 - p_1 p_4 p_2 + p_1 p_3 p_4 p_2$ . Note that

$$\begin{aligned} \mathbb{P}[\phi'] - \mathbb{P}[\phi] &= (p_1 p_2 p_3 p_4 - p_1 p_2 p_4)(p_4 - 1) \\ &= (p_1 p_2 p_4)(1 - p_3)(1 - p_4) \geq 0. \end{aligned}$$

**Lemma 42 (Positive DNF dissociation)** For every dissociation  $\phi'(\mathbf{x}')$  of a positive DNF  $\phi(\mathbf{x})$ , the following holds:  $\mathbb{P}[\phi'] \geq \mathbb{P}[\phi]$ .

*Proof (Lemma 42)* We will proceed in two steps. We first show that (a) the proposition holds for any single-step dissociations. We then (b) infer by induction for multi-step dissociations.

(a) *Single-step dissociation:* A single step dissociation is one where  $|\mathbf{x}'| = |\mathbf{x}| + 1$ , i.e. there is exactly one more variable appearing in the dissociation  $\phi'$  than  $\phi$ . We know that the substitution  $\theta$  must be surjective, i.e. each variable  $\in \mathbf{x}$  must be mapped to at least once, since all variables must appear at least once in  $\phi'(\theta(\mathbf{x}')) = \phi(\mathbf{x})$ . It follows that the size  $|\mathbf{x}'| \geq |\mathbf{x}|$ . It follows that a single-step dissociation is the simplest dissociation for which  $\phi$  and  $\phi'$  are not trivially isomorphic. From the pigeonhole principle, it also follows that there must be exactly two variables in  $\mathbf{x}'$  that are mapped to the same variable in  $\mathbf{x}$ . It also follows that the dissociation  $\phi'$  must have the same structure, i.e. that there must be a one-to-one mapping between conjuncts in  $\phi$  and  $\phi'$  with corresponding conjuncts containing the same number of variables, and that two variables that  $\theta$  maps to the same variable cannot appear in the same conjunct.

We assume w.l.o.g. that  $\theta(x_1) = \theta(x'_1) = x_1$  and  $\theta(x_i) = x_i, \forall i \in \{2, \dots, n\}$ . W.l.o.g., we further assume that  $x_1$  appears in the terms  $c_1, \dots, c_k$  ( $k \leq m$ ) of the DNF  $\phi$ , but not in  $c_{k+1}, \dots, c_m$ . W.l.o.g., we further consider a dissociation  $\phi'$  where  $x_1$  appears in the terms  $c_1, \dots, c_l$  ( $1 \leq l < k$ ) and  $x'_1$  in the terms  $c_{l+1}, \dots, c_k$ . We write  $c_j^*$  for  $c_j = x_1 c_j^*$ , ( $j \in \{1, \dots, k\}$ ), i.e. a conjunct  $c_1, \dots, c_k$  without the variable  $x_1$ . We can then write  $\phi(\mathbf{x})$  and  $\phi'(\mathbf{x}')$ , respectively, as

$$\begin{aligned} \phi(\mathbf{x}) &= \left(x_1 \wedge \bigvee_{j=1}^l c_j^*\right) \vee \left(x_1 \wedge \bigvee_{j=l+1}^k c_j^*\right) \vee \left(\bigvee_{j=k+1}^m c_j\right) \\ \phi'(\mathbf{x}') &= \left(x_1 \wedge \bigvee_{j=1}^l c_j^*\right) \vee \left(x'_1 \wedge \bigvee_{j=l+1}^k c_j^*\right) \vee \left(\bigvee_{j=k+1}^m c_j\right) \end{aligned}$$

with ( $1 \leq l < k \leq m$ ). Substituting the disjunctive expressions with  $D_i$ , we can write more compactly

$$\begin{aligned} \phi(\mathbf{x}) &= x_1 D_1 \vee x_1 D_2 \vee D_3 \\ \phi'(\mathbf{x}') &= x_1 D_1 \vee x'_1 D_2 \vee D_3 \end{aligned}$$

Using the inclusion-exclusion principle, we can write the event probabilities as

$$\begin{aligned} \mathbb{P}[\phi] &= p_1 \mathbb{P}[D_1] + p_1 \mathbb{P}[D_2] + \mathbb{P}[D_3] \\ &\quad - p_1 \mathbb{P}[D_1 D_2] - p_1 \mathbb{P}[D_1 D_3] - p_1 \mathbb{P}[D_2 D_3] + p_1 \mathbb{P}[D_1 D_2 D_3] \\ \mathbb{P}[\phi'] &= p_1 \mathbb{P}[D_1] + p_1 \mathbb{P}[D_2] + \mathbb{P}[D_3] \\ &\quad - p_1^2 \mathbb{P}[D_1 D_2] - p_1 \mathbb{P}[D_1 D_3] - p_1 \mathbb{P}[D_2 D_3] + p_1^2 \mathbb{P}[D_1 D_2 D_3] \end{aligned}$$

Comparing the two expressions, we get

$$\mathbb{P}[\phi'] - \mathbb{P}[\phi] = p_1(1 - p_1)(\mathbb{P}[D_1 D_2] - \mathbb{P}[D_1 D_2 D_3]) \geq 0 \quad (4)$$

since  $\mathbb{P}[\psi_1] \geq \mathbb{P}[\psi_1 \psi_2]$ .

(b) *Multi-step dissociation:* A  $k$ -step dissociation is one where single-step dissociations are consecutively applied ( $\phi \rightarrow \phi' \rightarrow \phi'' \rightarrow \dots \rightarrow \phi^{(k)}$ ). It trivially follows from transitivity that the proposition also holds for a  $k$ -step dissociation. It also follows that a  $k$ -step dissociation  $\phi^{(k)}$  has  $|\mathbf{x}^{(k)}| = |\mathbf{x}| + k$  variables.

Vice versa, every dissociation with  $|\mathbf{x}'| = |\mathbf{x}| + k$  can be constructed as a  $k$ -step dissociation as follows: Denote the number that a variable  $x_i \in \mathbf{x}$  is mapped to in  $\theta$  by  $k(i)$ . Then consider the following multi-step dissociation with  $k = \sum_{i=1}^n k(i) - 1$  steps: Iterate over all variables  $x_i$ . For each variable with has  $k(i) > 1$  dissociate the variable in  $k(i) - 1$  steps so that afterwards the substitution  $\theta(x_i) = \theta(x'_i) = \dots = \theta(x_i^{(k-1)}) = x_i$ . This can be done by partitioning the appearances of  $x_i$  in  $\phi$  according to the appearances of the variables in  $\phi'$  and dissociating these appearances one after the other. Hence, the proposition holds for any dissociation.  $\square$

Note that Equation 4 holds in any of 3 conditions:

- (i)  $p_1$  is either 0 or 1.
- (ii)  $D_1 = 0$  or  $D_2 = 0$ , which requires that in all conjuncts  $c_j^*$  of either  $D_1$  or  $D_2$  (written as  $D_{1/2}$ ) there must exist at least one variable with 0 probability:  $\forall c_j^* \in D_{1/2}. \exists i. x_i \in c_j^* : p_i = 0$ .
- (iii)  $D_3 = 1$ , which requires that there is at least one conjunct in which  $x_1$  does not appear that is 1:  $\exists c_j \in D_3. \forall i. x_i \in c_j : p_i = 1$ .

## C.2 Actual proof of Theorem 12 (Partial dissociation order)

*Proof (Theorem 12: Partial dissociation order)* (a) We have to show both directions and start with

$$\Delta \succeq \Delta' \Rightarrow \forall D : r(q^\Delta) \geq r(q^{\Delta'})$$

i.e. whenever two dissociations are comparable, then we know which has a higher reliability scores on every database instance  $D$ . This follows from Lemma 42 and the observation that there is a one-to-one correspondence between a query dissociations and a positive DNF dissociation. Concretely, at the level of its *lineage expression*, a query dissociation is a dissociation of a  $k$ -partite DNF.

(b) We next prove

$$\Delta \succeq \Delta' \iff \forall D : r(q^\Delta) \geq r(q^{\Delta'})$$

i.e. whenever  $r(q^\Delta) \geq r(q^{\Delta'})$  holds for two dissociations over any database instance  $D$ , then  $\Delta \succeq \Delta'$  in the partial dissociation order. In other words, for any two dissociations  $\Delta$  and  $\Delta'$  of a query  $q$  that are incomparable, i.e.  $\Delta \not\succeq \Delta'$  and  $\Delta' \not\succeq \Delta$ , there exist two database instances so that the dissociated probability of either dissociation becomes bigger. We prove this by showing the contrapositive

$$\Delta \not\succeq \Delta' \implies \exists D : r(q^\Delta) < r(q^{\Delta'})$$

i.e. for any two dissociations  $\Delta$  and  $\Delta'$  of  $q$ , with  $\Delta \not\succeq \Delta'$ , there exists a database instance so that  $r(q^\Delta) < r(q^{\Delta'})$ . We will achieve this by constructing a database instance  $D$  so that  $r(q^\Delta) = r(q)$ , but  $r(q^{\Delta'}) > r(q)$ : Since  $\Delta \not\succeq \Delta'$ , there must exist one variable  $x_i \in \mathbf{y}_j$  that is dissociated in relation  $R_j$  of  $\Delta'$  which is not dissociated in  $\Delta$ . W.l.o.g. let  $x_1 \in \mathbf{Var}(q)$  be this variable, and  $R_1$  be the relation. W.l.o.g. consider the active domain  $A$  of the database to be  $\{a, b\}$ . Then construct the following database instance  $D$ :

1. For each relation  $R_i \neq R_1$  with  $x_1 \notin \mathbf{Var}(R_i)$ , insert one deterministic tuple ( $p = 1$ ) with 'a' as value for each attribute  $x_i \in \mathbf{Var}(R_i)$ :

$$R_i(a, a, \dots, a), p = 1$$

2. For each relation  $R_i \neq R_1$  with  $x_1 \in \mathbf{Var}(R_i)$ , insert two deterministic tuples ( $p = 1$ ) with 'a' as value for each attribute  $x_i \neq x_1$ , and either 'a' or 'b' as value for attribute  $x_1$ :

$$R_i(a, a, \dots, a), p = 1$$

$$R_i(b, a, \dots, a), p = 1$$

3. For relation  $R_1$  with  $x_1 \notin \mathbf{Var}(R_1)$ , insert one uncertain tuple ( $p = 0.5$ ) with 'a' as value for each attribute:

$$R_1(a, a, \dots, a), p = 0.5$$

Then for every dissociation  $\Delta$  for which  $x_1 \notin \mathbf{y}_1$ :  $r(q^\Delta) = r(q) = 0.5$ . This follows from two facts: (i) certain tuples that get dissociated do not change the query probability; (ii) the only probabilistic tuple in  $R_1$  only gets dissociated if the dissociation includes  $x_1$  since all other variables only include one single value in the active domain. On the other hand, for every dissociation  $\Delta'$  for which  $x_1 \in \mathbf{y}_1$ :  $r(q^{\Delta'}) = 0.75 > r(q)$ . Hence, we have shown that for  $D$ :  $r(q^\Delta) < r(q^{\Delta'})$ .  $\square$

### D Section 3: Proof Proposition 17

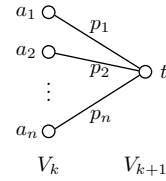
*Proof (Prop. 17: connection to networks)* Note that we use digraphs to enforce that each path from  $s$  to  $t$  has exactly  $k$  edges.

(a) We first establish the connection between *graph reliability* and *query reliability*. The first claim is obvious: a possible world contains a path from  $s$  to  $t$  iff the query is true on that world: The chain query is true exactly if, in a randomly chosen world, there is a set of tuples of each relation that forms at least one output tuple. This corresponds exactly to the graph reliability, i.e. the probability that the nodes  $s$  and  $t$  are connected in a randomly chosen subgraph.

(b) We next establish the connection between *propagation in networks* and *dissociation in databases*. Consider the unique safe query plan for the dissociated query  $q^\Delta$ :

$$P = \pi_{x_k}^p \bowtie^p [R_k^0(x_k, t), \pi_{x_{k-1}}^p \bowtie^p [R_{k-1}^0(\mathbf{x}_{[k-1, k]}), \dots, \pi_{x_2}^p \bowtie^p [R_1^{x_{[3, k]}(s, \mathbf{x}_{[2, k]}), R_2^{x_{[4, k]}(\mathbf{x}_{[2, k]})}]]]]$$

This plan is evaluated from the inside out. The table  $R_1$  is dissociated on all variables except  $x_2$ , i.e. each consequent project on previous join results from  $R_1$  will treat each tuple as independent. The independent project corresponds exactly to the way propagation is calculated at each node iteratively from the probabilities of its parents and incoming edges. We prove this by induction on  $k$ : When  $k = 1$  then  $R_1$  contains a single edge  $(s, t)$ , whose probability is equal to  $r(q)$ , to  $r(q^\Delta)$ , and to the network propagation score. To prove for  $k \geq 1$ , let  $V_k = \{a_1, \dots, a_n\}$  be the nodes in the before-last partition (the last partition is  $V_{k+1} = \{t\}$ ).



We defined in Example 1 the network propagation score to be:

$$\rho(t) = 1 - \prod_i (1 - \rho(a_i) \cdot p_i)$$

where  $p_i$  is the probability of the edge  $(a_i, t)$ . On the other hand, the reliability of the dissociated query is given by the following formula which represents a probabilistic join with  $R_k(x_k, t)$ , followed by a probabilistic projection on the variable  $x_k$ , and where an expression  $[a/x]$  stands for substitution of a variable  $x$  by a constant  $a$ :

$$\begin{aligned} r(q^\Delta) &= 1 - \prod_i (1 - r(q^\Delta[a_i/x_k])) \\ &= 1 - \prod_i (1 - r(R_1(s, \mathbf{x}_{[2, k-1]}, a_i), \dots, R_{k-1}(x_{k-1}, a_i)) \cdot r(R_k(a_i, t))) \\ &= 1 - \prod_i (1 - \rho(a_i) \cdot p_i) \quad \square \end{aligned}$$

### E Section 4: Proof Theorem 18

*Proof (Theorem 18: Safe dissociation)* The isomorphism is as follows: we go from a safe dissociation  $\Delta$  to the corresponding plan  $P = f(\Delta)$  by taking the unique plan  $P^\Delta$  and dropping all dissociated variables from the relations. 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. We go from a plan  $P$  to a safe 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  $\mathbf{JVar} - \mathbf{HVar}(P_j)$ . To prove the isomorphism, we have to show both directions:

(a)  $g(f(\Delta)) = \Delta$ : Consider a safe dissociation  $\Delta$  and denote its corresponding unique safe plan  $P^\Delta$ . This plan uses dissociated relations, hence each relation  $R_i^{y_i}(x_i, \mathbf{y}_i)$  has possibly some extraneous variables  $\mathbf{y}_i$ . If we drop all variables  $\mathbf{y}_i$  from the relations, then this transforms  $P^\Delta$  into a regular (unsafe) plan  $P$  for  $q$ . If we now consider all those variables  $\mathbf{y}_i$  that we have thus dropped from a relation  $R_i$ , then these are exactly those variables that are added by recursively adding all existential variables  $\mathbf{JVar} - \mathbf{HVar}(P_j)$  for each join operator.

(b)  $f(g(P)) = P$ : Consider a possibly unsafe plan  $P$  for  $q$  and recursively dissociate 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  $\text{JVar} - \text{EVar}(P_j)$ . Then this defines a unique dissociation  $\Delta$  of  $q$ . Now consider the unique safe plan for  $q^\Delta$ . Since the safe plan  $P^\Delta$  is the only plan for which all subplans share the same head variables, it must be the same plan as the original unsafe plan for  $q$ .  $\square$

## F Section 4: Proof Algorithm 1

*Proof (Theorem 22: Algorithm 1)* First, recall from the proof of Theorem 18 that we go from a plan  $P$  to a safe dissociation 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  $\text{JVar} - \text{HVar}(P_j)$ . Hence, in a project plan  $P = \pi_{\mathbf{x}}^P P'$ , all relations occurring in  $P'$  either already contain or are dissociated on each variable in  $\mathbf{x}$ . This is since  $\mathbf{x}$  are the join variables if  $P'$  is a join plan, and it holds trivially if  $P'$  is a single relation.

(1) *Soundness:* We show that every plan produced by Algorithm 1 corresponds to a minimal safe dissociation, i.e. it is not dominated by (that means always bigger than) any other plan. We show this by induction on the set of relations in a plan. At each step, any query must either project or join. Joins are only possible if the join variables are identical to the head variables of the plan, hence if the subplans share no existential variables. If at a step, there are disconnected components, then all minimal plans need to join on those components (as any project on a set of variables is clearly dominated by a join), then project on those variables only in the component that contains this set. If the relations are connected, then each plan needs to project on variables so that the query becomes disconnected. We need to show that projecting on a minimal set of variables that disconnects the query cannot be dominated by any other safe dissociation. This follows immediately, as any projection on a subset does not disconnect the query, and every projection on a superset is dominated, and any overlapping set of variables cannot dominate this dissociation.

(2) *Completeness:* We show that Algorithm 1 returns a plan for each minimal safe dissociation. We again show this by induction on the set of relations in a plan. Recall from before that if a query is disconnected, then the minimal query plan needs to join all components and we only need to focus on the case when the query is not disconnected. We need to consider all possible subsets of the variables that disconnect the query. We have shown that only those can be minimal that are not supersets of variables that alone disconnect. Since the algorithm iterates over all minimal subsets that disconnect the query, it is complete.  $\square$

## G Section 4: Proof Proposition 23

In the following, we write  $\mathcal{P}_{>k}([n])$  for the set of subsets of the powerset  $\mathcal{P}([n])$  of cardinality bigger or equal to  $k$ .

**Lemma 43 (DNF polynomial)** *Let  $\phi(\mathbf{x})$  be a positive minimal  $k$ -uniform DNF of size  $m$  containing  $n$  total variables. Then the multilinear polynomial for  $\phi(\mathbf{x})$  is*

$$\mathbb{P}[\phi(\mathbf{x})] = \sum_{j=i \in \mathbf{c}_j} p_i + \sum_{S \in \mathcal{P}_{>k}([n])} c(S) \prod_{i \in S} p_i$$

In other words, all of the terms of the multilinear polynomial for  $\phi$  are of order bigger or equal to  $k$ . Furthermore, most are of order  $> k$  except for  $\sum_{j=i \in \mathbf{c}_j} p_i$ , which corresponds to summing up the probabilities of each conjunct.

*Proof (Lemma 43)* Lemma 43 follows immediately from Inclusion-Exclusion (Equation 3) and noting that any conjunction of terms  $c_{j_1} \wedge$

$c_{j_2}$  needs to consist of at least  $k+1$  factors. This follows from the fact that any two terms  $c_{j_1} \neq c_{j_2}$  need to have at least one different variable. For example,  $x_1 x_2 x_3 \wedge x_1 x_2 x_4 = x_1 x_2 x_3 x_4$ .

*Proof (Prop. 23)* The lineage for a self-join-free conjunctive query  $q$  is a positive  $k$ -uniform DNF of size  $m$ . Also, the lineage of any dissociation  $q'$  of this query (in particular the one with the minimal score) is a positive  $k$ -uniform DNF of size  $m$ . It follows that

$$\begin{aligned} \mathbb{P}[q] &= \sum_{j=i \in \mathbf{c}_j} p_i + \sum_{S \in \mathcal{P}_{>k}([n])} c(S) \prod_{i \in S} p_i \\ \mathbb{P}[q'] &= \sum_{j=i \in \mathbf{c}_j} p_i + \sum_{S \in \mathcal{P}_{>k}([n])} c'(S) \prod_{i \in S} p_i \end{aligned}$$

Next consider the operation of scaling down all probabilities by  $f$ ,  $p'_i = f p_i$ , and the implication on  $\varepsilon := \frac{\mathbb{P}(q) - r(q)}{r(q)} = \frac{\mathbb{P}(q') - \mathbb{P}[q]}{\mathbb{P}[q]}$ . Observe that in the nominator, the terms of order  $k$  cancel out and it becomes a sum of terms with minimum order  $k+1$ . In contrast, the denominator still has minimum order of  $k$ . After dividing both nominator and denominator by  $f^k$ , we have the each term in the denominator is multiplied with at least  $f$ , whereas the denominator has some terms without  $f$ . We thus have  $\lim_{f \rightarrow 0^+} \varepsilon = 0$ .

## H Section 4: Number of minimal query plans

*Star queries.* Consider the  $k$ -star query  $q: -R_1(x_1), \dots, R_k(x_k), U(\mathbf{x})$  and the  $k!$  permutations on the order on  $\mathbf{x}$ . To simplify notation and w.l.o.g., we consider the permutation  $\sigma = (x_1, \dots, x_k)$ . We show the following query plan is minimal:

$$P = \pi_{x_1}^P \bowtie^P [R_1(x_1), \pi_{x_2}^P \bowtie^P [\dots, \pi_{x_k}^P \bowtie^P [R_k(x_k), U(\mathbf{x})]]]$$

$P$  has  $k$  projections and corresponds to a dissociation  $\Delta$ , where relation  $R_i$  is dissociated on  $i-1$  variables  $\mathbf{x}_{[1,i-1]}$ . Each query plan according to Definition 3 must have either one projection and a single relation, or a join between more relations at the end of its branches. No query plan for  $q$  can have a projection at the leaf since each variable appears in at least 2 relations. Since there is only one relation  $U$  that joins with other relations, there can only be one leaf with a join between  $U$  and at least one other relation  $R_i$  and there cannot be several branches. Further any query plan for  $q$  can have at maximum  $k$  projections for each of the  $k$  variables. As a consequence, each plan with  $k$  projections is isomorphic to a total order on  $\mathbf{x}$ . Next consider a query plan that has less than  $k$  projections while keeping the order of  $\sigma$ . This plan must have at least one  $R_i$  that is dissociated on at least  $i$  variables and is, hence, dominated by  $P$ . Hence,  $P$  is minimal. Since there are  $k!$  possible permutations, there are  $k!$  such minimal plans.

The total number of plans (or safe dissociations) is equal to the number of weak orderings definable on  $k$  alternatives. This is a consequence of the relation  $U(\mathbf{x})$  which contains all variables. Hence, each query plan must define a hierarchy between the variables in which all variables are comparable, and in which ties are allowed. This is exactly the definition of a weak ordering (or total preorder), and the number of such is given by the OEIS sequence A000670<sup>12</sup>: 1, 3, 13, 75, 541, 4683, 47293, 545835, ...

*Chain queries.* Next consider the Boolean  $k+1$ -chain query  $q: -R_1(x_1), R_2(x_1, x_2), \dots, R_{k+1}(x_k)$ . Note that each variable is shared only by two subsequent relations. The leaves of each query plan must thus have exactly two relations. Furthermore, every minimal query plan must have exactly  $k$  joins between subplan corresponding to combining one variable at a time. Hence, the number of minimal query plans for a  $k+1$ -chain query is equal to the number of ways to insert  $k$  pairs

<sup>12</sup> <http://oeis.org/classic/A000670>

of parentheses in a word of  $k+1$  letters. This corresponds to OEIS sequence A000108<sup>13</sup>: 1, 2, 5, 14, 42, 132, 429, 1430, ...

The number of total query plans corresponds to OEIS sequence A001003<sup>14</sup>: 1, 3, 11, 45, 197, 903, 4279, 20793, ... This is the number of ways to insert parentheses in a string of  $k$  symbols. The parentheses must be balanced but there is no restriction on the number of pairs of parentheses. The number of letters inside a pair of parentheses must be at least 2. Parentheses enclosing the whole string are ignored.

The total number of dissociations for both,  $k$ -star query and  $k+1$ -chain query, is given by  $2^K$ , where  $K = k(k-1)$  is the number of undissociated variables. Figure 8 summarizes these numbers in one table.

## I Section 5.1: Deterministic Tables

*Proof (Lemma 26: deterministic table dissociation)* This lemma follows immediately from Equation 4 in the proof for Lemma 42 in Appendix C: Whenever a positive DNF expression  $\phi$  gets dissociated only on variables with probability 1 into expression  $\phi'$ , then their probabilities are the same:  $\mathbb{P}[\phi] = \mathbb{P}[\phi']$ . Adding variables to deterministic tables corresponds to dissociating only deterministic tuples with probability 1. Hence, the original and the dissociated query have the same query reliability.  $\square$

**Lemma 44 (Deleting variables from safe queries)** *If  $q$  is safe and  $x \in \text{EVar}(q)$ , then deleting  $x$  from  $q$  leads to a query  $q'$  that is still safe.*

*Proof (Lemma 44)*  $q$  is safe iff for any two existential variables  $y, z$ , one of the following three conditions hold:  $sg(y) \subseteq sg(z)$ ,  $sg(y) \cap sg(z) = \emptyset$ , or  $sg(y) \supseteq sg(z)$ . Deleting another variable  $x$  from the query does not change the hierarchy between the other variables.  $\square$

*Proof (Lemma 28)* Let  $q$  be connected by its existential variables and  $x \in \text{SVar}(q)$ . If  $x$  is contained in all subgoals, then  $x$  is a root variable. If  $x$  is not contained in all subgoals, then there exists a top set  $\mathbf{y}$  considered by Algorithm 1 that does not contain  $x$ . We need to show that, for any safe dissociation  $\Delta_{\mathbf{y}}$  with  $\mathbf{y}$  as root variables, there is another safe dissociation  $\Delta_x$  with  $x$  as root variable and  $r(q^{\Delta_x}) \leq r(q^{\Delta_{\mathbf{y}}})$ .

For that, consider the dissociation  $\Delta_{\mathbf{y}x}$  obtained by further dissociating  $\Delta_{\mathbf{y}}$  on  $x$ . According to Lemma 26,  $r(q^{\Delta_{\mathbf{y}x}}) = r(q^{\Delta_{\mathbf{y}}})$ , and according to Lemma 44,  $\Delta_{\mathbf{y}x}$  must be safe. As a consequence,  $\Delta_{\mathbf{y}x}$  is a safe dissociation that has  $\mathbf{y} \cup x$  as root variables. Therefore, there exists a minimal safe dissociation  $\Delta_x$  (which may or may not be the same as  $\Delta_{\mathbf{y}x}$ ) with  $x$  as root variable and  $r(q^{\Delta_x}) \leq r(q^{\Delta_{\mathbf{y}x}}) = r(q^{\Delta_{\mathbf{y}}})$ . As a consequence, minimal query plans with  $x$  as root variable will always have a lower or equal score as plans with  $\mathbf{y}$  as root variables. Hence, all minimal plans have  $x$  as root variable.  $\square$

*Proof (Theorem 29) Soundness:* We show that every plan produced by Algorithm 2 corresponds to a minimal safe dissociation, i.e. there are no two plans produced by Algorithm 2 where one plan dominates the other. Note that Algorithm 2 can only create two different plans in line 16, i.e. whenever we don't have a separator variable and need to iterate over all top sets. For each two different top sets, the algorithm needs to dissociate at least one non-deterministic relation on a variable that the other one does not dissociate on. Hence no query plan produced by Algorithm 2 can dominate another one.

*Completeness:* We show that Algorithm 2 returns a plan for each minimal safe dissociation, i.e. for every plan  $P$  from Algorithm 1, there is a plan  $P$  from Algorithm 2 with lower or equal score. If  $q$  is disconnected, then nothing changes from Algorithm 1, and we only need to focus on the case when the query is not disconnected. If  $q$  is connected, then completeness follows immediately from Lemma 28: all minimal plans need to have all separator variables as root variables.  $\square$

<sup>13</sup> <http://oeis.org/classic/A000108>

<sup>14</sup> <http://oeis.org/classic/A001003>

$R$	$A$	$S^d$	$A$	$C$	$T^d$	$B$	$C$	$U$	$B$
$r_1$	$a$	$s_1$	$a$	$c$	$d_1$	$e$	$c$	$t_1$	$e$
$r_2$	$b$	$s_2$	$b$	$c$	$d_2$	$f$	$c$	$t_2$	$f$

(a)  $D$

$R$	$x$	$y$	$z$	$R$	$x$	$y$	$z$	$R$	$x$	$y$	$z$	$R$	$x$	$y$	$z$
$U$	$\circ$	$\circ$	$\circ$	$U$	$\circ$	$\bullet$	$\bullet$	$U$	$\circ$	$\circ$	$\circ$	$U$	$\circ$	$\circ$	$\circ$
$S^d$	$\circ$	$\circ$	$\circ$	$S^d$	$\circ$	$\circ$	$\circ$	$N^d$	$\circ$	$\circ$	$\circ$	$N^{sd}$	$\circ$	$\circ$	$\circ$
$T^d$	$\circ$	$\circ$	$\circ$	$T^d$	$\circ$	$\circ$	$\circ$								

(b)  $q$       (c)  $q^{\Delta}$       (d)  $q'$       (e)  $q''$

**Fig. 24** Example 45: Query  $q: -R(x), S^d(x, z), T^d(y, z), U(y)$  and minimal safe dissociation  $q^{\Delta}$  that calculates  $r(q)$  exactly on database instance  $D$ . (d,e): First joining both deterministic tables into  $N^d = \bowtie [S^d, T^d]$  and projecting  $z$  away does not change the query reliability. However, there is no minimal safe dissociation anymore that allows to calculate the original reliability exactly:  $\rho(q'') > \rho(q)$ .

### I.1 Remark on completeness for deterministic relations

We give an example, similar in spirit to Example 27, that illustrates that the sound and complete enumeration in the presence of deterministic relations is non-trivial. A reasonable idea would be to first join all deterministic relations, thereby eagerly reducing variables that appear only in deterministic relations. However, this process would not lead to completeness, as we will illustrate with a counter-example.

*Example 45 (Incorrect deterministic dissociation 2)* Consider the query  $q: -R(x), S^d(x, z), T^d(y, z), U(y)$  with two deterministic relation  $S^d$  and  $T^d$  with incidence matrix Fig. 24b over the database instance  $D_2$  from Fig. 24a. Then lineage of  $q$  is again  $\text{Lin}(q) = r_1 s_1 t_1 u_1 \vee r_1 s_1 t_2 u_2 \vee r_2 s_2 t_1 u_1 \vee r_2 s_2 t_2 u_2$ . Replacing  $s_1, s_2, t_1$  and  $t_2$  with 1, the lineage can be simplified and factored into  $\text{Lin}(q) = (r_1 \vee r_2)(u_1 \vee u_2)$ , which is a read-once formula. Assuming all non-deterministic tuples to have the same probability 0.5, the query reliability can easily be calculated as  $\mathbb{P}[q] = 9/16 \approx 0.563$ . Dissociating  $q$  on  $z$  (Fig. 24c) results in a safe dissociation  $q^{\Delta}$  that turns out to have, on the particular database instance  $D_2$ , exactly the same lineage expression as the original query:  $\text{Lin}(q^{\Delta}) = \text{Lin}(q)$ . Since it is a safe dissociation, there is a unique plan  $P(q^{\Delta})$  that calculates the reliability of  $q^{\Delta}$ , and its score is exactly the reliability of the original query:  $\rho(q) = \text{score}(P(q^{\Delta})) = r(q) \approx 0.563$ . However, if we had first joined the two deterministic relation  $S^d$  and  $T^d$  into  $N^d(x, y, z) = \bowtie [S^d(x, z), T^d(y, z)]$  (Fig. 24d), and then projected the variable  $z$  away into  $N^{sd} = \pi_{-z} N^d(x, y, z)$ , then the propagation score of  $q''$  turns out to be bigger than the one of the original query:  $\rho(q'') = 39/64 \approx 0.609 > r(q'') = \rho(q)$ .

## J Section 5.2: Functional Dependencies

*Proof (Lemma 31: FD dissociation and reliability)* Assume that  $\Gamma: \mathbf{x} \rightarrow \mathbf{y}$  holds on relation  $R_i$ , and that there exists another relation  $R_j$  with  $\mathbf{x} \subseteq \text{Var}(R_j)$ , but  $\mathbf{y} \notin \text{Var}(R_j)$ . Then  $\Gamma$  also holds for the natural join between  $R_i$  and  $R_j$ . Furthermore, any join result with the same values  $\mathbf{a}$  for  $\mathbf{x}$  has exactly one tuple from  $R_j$  in its lineage, which does not change after dissociating  $R_j$  on  $\mathbf{y}$ . Therefore, also the lineage and the query reliability remain the same.  $\square$

*Proof (Lemma 32: FD dissociation and hierarchies)* Assume that the FD  $\Gamma: \mathbf{x} \rightarrow \mathbf{y}$  holds and that query  $q$  is safe and thus hierarchical. We show that applying  $\Gamma$  eagerly to all relations in  $q$  results in a query  $q^{\Delta}$  that is still hierarchical. Let  $sg(\mathbf{x})$  denote the subgoals containing all

variables of  $\mathbf{x}$ ,  $sg(y)$  the subgoals containing  $y$  in  $q$ , and  $sg(\bar{y})$  the subgoals containing  $y$  or  $\bar{y}$  in  $q^A$ . First note that  $sg(\mathbf{x}) \cap sg(y) \neq \emptyset$  since we have one relation that enforces  $\Gamma$ . If  $sg(\mathbf{x}) \supseteq sg(y)$ , then  $sg(\mathbf{x}) = sg(\bar{y})$  and  $q^A$  is still hierarchical, as removing  $y$  from the original hierarchy cannot invalidate the hierarchy. If, however,  $sg(\mathbf{x}) \subset sg(y)$ , then  $sg(y) = sg(\bar{y})$  and thus the hierarchy remains unchanged.  $\square$

*Proof (Theorem 34: Algorithm 3) (1) Soundness:* We show that every plan  $P_1$  produced by Algorithm 3 corresponds to a minimal safe dissociation, i.e. there is no other plan  $P_2$  produced by Algorithm 3 that strictly dominates  $P_1$ . This follows immediately from Algorithm 2: let  $\tilde{q}$  be the query dissociated on all functional dependencies. If  $\tilde{q}$  is safe, then there is just one query plan. If  $\tilde{q}$  is not safe, then there are at least two different top sets, either of which dissociates at least one relation on a variable that the other one does not. Hence no query plan produced by Algorithm 3 can strictly dominate another one.

(2) *Completeness:* We show that Algorithm 3 returns a plan for each minimal safe dissociation, i.e. for every plan  $P'$  produced by Algorithm 2, there is one plan  $P$  produced by Algorithm 3 that dominates  $P'$ . Suppose the contrary and let  $P'$  be such a plan that dominates all plans produced by Algorithm 2. Then let  $q'$  be the dissociated query corresponding to  $P'$  and let  $q''$  be the query after applying the eager FD dissociation to  $q'$ . Then, we know from Lemma 31 that  $r(q'') = r(q')$ . Furthermore, since  $q'$  is safe, we know from Lemma 32 that  $q''$  must be safe as well. Furthermore, we know that  $q'' \succeq \tilde{q}$  in the partial dissociation order of  $\tilde{q}$ . As a consequence, Algorithm 3 will enumerate a query plan  $P''$  with  $score(P'') = r(q'') = r(q')$  in one of the produced minimal query plan which violates our assumption.  $\square$