Query Processing over Uncertain Data

Nilesh Dalvi
Troo.ly Inc.

Dan Olteanu
University of Oxford

SYNONYMS

Query Processing over Probabilistic Data

DEFINITION

An uncertain or probabilistic database is defined as a probability distribution over a set of deterministic database instances called *possible worlds*.

In the classical deterministic setting, the query processing problem is to compute the set of tuples representing the answer of a given query on a given database. In the probabilistic setting, this problem becomes the computation of all pairs (t, p), where the tuple t is in the query answer in some random world of the input probabilistic database with probability p.

SCIENTIFIC FUNDAMENTALS

Representation of Uncertain Data

All aspects of query processing over uncertain data, and in particular its complexity and existing techniques, highly depend on data representation. Since it is prohibitively expensive to explicitly represent the extremely large set of all possible worlds of a probabilistic database, one has to settle for succinct data representations. Three such representations of increasing expressiveness are discussed next [34].

In *tuple-independent (TI) databases*, the tuples are independent probabilistic events. A TI database of n tuples represents the probabilistic database that is the powerset of the input set of n tuples and thus has 2^n possible worlds. In *block-independent-disjoint (BID) databases*, the tuples are partitioned into blocks such that tuples within the same block are disjoint events and tuples from different blocks are independent. A BID database represents all possible worlds with at most one tuple from each block. In *probabilistic conditional (PC) databases*, the tuples are associated with propositional formulas over independent random variables. Each total assignment ψ of these random variables defines a world consisting of those tuples whose formulas are satisfied by ψ. The

probability of the world is the product of the probabilities of the assignments of the random variables in ψ. A TI databases is a PC database where each tuple is associated with a distinct Boolean random variable. Also, a BID database is a PC database where each relation is made up of blocks of tuples such that the formulas of the tuples within a block are mutually exclusive and the formulas of the tuples from different blocks are over disjoint sets of variables.

PC databases can represent the answers of any relational query over PC databases. The formula associated with an answer tuple t is called the lineage of t and explains which tuples must be present in the input database in order for t to be in the query answer. Given two tuples with formulas ϕ_1 and ϕ_2, their join is a tuple with lineage $\phi_1 \wedge \phi_2$, their union is a tuple with lineage $\phi_1 \vee \phi_2$, and their difference is a tuple with lineage $\phi_1 \wedge \neg\phi_2$.

In contrast to BID and TI databases, PC databases are complete in the sense that they can represent any probabilistic database. However, the formalism defined by conjunctive queries over BIDs is complete. Despite their restricted nature, TI databases are practical; for instance, inference in Markov Logic Networks can be reduced to relational query processing over TI databases.

Prime examples of TI databases are the Google Knowledge Vault [8] and NELL (Never Ending Language Learner, `http://rtw.ml.cmu.edu/rtw/`) knowledge bases, of BID databases are the Google Squared tables [10], and of PC databases are the answers and their lineage for relational queries on probabilistic databases in any of the above formalisms.

Complexity of Query Processing

The *data complexity* of queries over probabilistic databases represented as TI, BID, or PC databases is #P-hard. This high computational complexity is already witnessed for simple join queries on TI databases, since the computation of marginal probabilities may require to enumerate all possible worlds. Several classes of relational queries, e.g., non-repeating conjunctive queries [5], their ranking version [27] and extension with negation [12], unions of conjunctive queries [7], and aggregate queries [30] exhibit an interesting dichotomy property: the data complexity of every query in any of these classes over TI databases is either polynomial time or #P-hard. Beyond relational queries and TI databases, hardness and tractability results have been shown, e.g., for XML queries over Recursive Markov Chains and their restrictions [3, 24], for graph queries over probabilistic RDF graphs [25], for event queries over Markovian streams [29], and for conjunctive queries over graphical models [31].

Query Processing Techniques

To overcome the high complexity of probabilistic query processing, two avenues of research have been pursued [34]. The first avenue is to identify tractable queries, i.e., queries computable in polynomial time, and develop efficient processing techniques for this subset. Techniques for tractable queries developed

in the context of the MystiQ [5] and SPROUT [11] probabilistic database systems show performance for tractable queries over probabilistic databases close to that for queries over classical deterministic databases. The second avenue is to develop techniques that approximate the probabilities of the query answers. In many applications, probabilities are only needed to rank the query answers and approximate probabilities may suffice for ranking. Even when probabilities are returned to the user, it is often desirable to improve performance by sacrificing precision. In addition to MystiQ and SPROUT, Trio [36], Orion [32], and MayBMS [17] also employ approximate techniques for hard queries.

An orthogonal classification considers whether the probabilistic inference task is performed inside or outside the database engine. In-database techniques cast the inference task as query processing in deterministic databases. Out-database techniques rely on specialized inference methods beyond the capabilities of a database engine, e.g., knowledge compilation techniques. Prime examples of systems using out-database techniques are Trio, MayBMS, and PrDB [31], whereas MystiQ, Orion, and MCDB use in-database techniques; SPROUT uses in-database techniques for classes of tractable queries and out-database techniques for hard queries. The two types of techniques can also be combined: In-database approaches are first applied to tractable subproblems and then out-database approaches are used for the remaining hard subproblems.

We next highlight several techniques for relational query processing over probabilistic databases. We leave aside processing techniques for complex queries such as nearest neighbors and skyline queries as they represent a topic on their own [4]. We also only refer to a few approaches to probabilistic ranking and refer the reader to a recent monograph for in-depth treatment [18].

In-database techniques

In-database techniques are used for exact evaluation of tractable queries in tuple-independent probabilistic databases, e.g., using safe plans [6] as discussed below, and also for approximate evaluation of hard queries, e.g., computing lower and upper bounds on answer probabilities via dissociation of input probabilistic events [14] or running Monte Carlo simulations that aggregate the query answers over several possible worlds sampled from complex probabilistic models [20].

Such techniques extend standard operators in relational plans to compute both tuples and their probabilities. Examples of extended operators are the independent join operator, which multiplies the probabilities of the tuples it joins under the assumption that they are independent, and the independent project operator, which computes the probability of an output tuple t as $1 - (1 - p_1) \cdots (1 - p_n)$ where $p_1, \ldots, p_n$ are the probabilities of all tuples that are input to the operator and project into t, again assuming that these tuples are independent. The selection operator retains the tuples that satisfy its condition, along with their probabilities.

For a given relational query, not all of its query plans with extended operators compute the probabilities correctly since the independence assumption for the input to these operators may not hold. If the plan does compute the

probabilities correctly for any input database, then it is called a *safe query plan*. Safe plans are easily added to a relational database engine, either by small modifications to the relational operators or even without any change in the engine by simply rewriting the SQL query to add aggregates that manipulate the probabilities explicitly. If a query admits a safe plan, then its data complexity is polynomial time because any safe plan can be computed in polynomial time in the size of the input database by simply evaluating its operators bottom-up. Consequently, #P-hard queries cannot admit a safe plan. For the class of non-repeating conjunctive (select-project-join) queries, the selection, independent join, and independent project operators are complete since the tractable queries are precisely those that admit safe plans [5]. For TI databases under functional dependencies, hard queries may become tractable and computable using safe plans [26].

A variant of this technique is to decouple the computation of answer tuples from the computation of their probabilities, and to use different query plans for the two computation tasks. This is motivated by the observation that safe plans, while necessary for correct probability computation, can be suboptimal for computing the answer tuples [26]. This approach first computes the answer tuples and a relational encoding of their lineage using an optimized query plan, and then it uses the safe plan over the lineage to compute the probabilities of the answer tuples.

Safe plans can be extended to cope with richer classes of queries. Plans for tractable unions of conjunctive query terms use an inclusion-exclusion (IE) operator to compute their probabilities as a function of the probabilities of conjunctions of subsets of the query terms [7]. Safe plans with an additional disjoint project operator can compute non-repeating conjunctive queries over BID databases [28]. This operator sums up the probabilities of all the input tuples that project into the same output tuple.

Out-database techniques

Out-database query processing techniques are more general than in-database ones. They work for arbitrary relational queries and probabilistic data representations beyond TI databases.

An important class of out-database techniques draws on connections between probabilistic query processing and knowledge compilation. They compile the lineage of answer tuples into decision diagrams, e.g., Ordered Binary Decision Diagrams (OBDDs) and Deterministic Decomposable Negation Normal Forms (d-DNNFs), that admit linear-time probability computation. While in general the compilation can take time exponential in the number of random variables in the lineage, it takes polynomial time for several known classes of tractable queries on tuple-independent databases, e.g., the tractable non-repeating queries with negation [12], the non-repeating inversion-free unions of conjunctive queries [34], and queries with inequalities [21].

Recall that lineage is a propositional formula Φ over Boolean[1] random variables x_1 to x_n. The idea behind lineage compilation is to recursively apply the following two steps in the given order:

1. If $\Phi = \psi_1 \vee \cdots \vee \psi_m$ or $\Phi = \psi_1 \wedge \cdots \wedge \psi_m$ such that ψ_1 to ψ_m are pairwise independent or mutually exclusive, then the probability of Φ can be computed from the probabilities of ψ_1 to ψ_m in linear time. This step precisely captures the operators independent/disjoint project/join operators used by safe plans.

2. If the previous step is not applicable (such as for hard queries), then we apply *Shannon expansion* (DPLL) on any variable x_i in Φ: Φ is equivalent to a disjunction of two mutually-exclusive expressions $x_i \wedge \Phi_{x_i} \vee \neg x_i \wedge \Phi_{\neg x_i}$, where Φ_{x_i} and $\Phi_{\neg x_i}$ are Φ where x_i is set to true and false, respectively. The probability of Φ is the sum of the probabilities of the two mutually-exclusive formulas, where Φ_{x_i} and $\Phi_{\neg x_i}$ have at least one variable (x_i) less than Φ.

We exhaustively repeat these two steps until we reach the Boolean constants true or false. Since at each compilation step we can compute in linear time the probability of the formula using the probabilities of the child subformulas, the algorithm runs in time linear in the number of steps. The order of variables x_i in Shannon expansion steps drastically influences the number of compilation steps, which can be at most exponential in the number of variables [34].

If the compilation is stopped at any time before reaching a Boolean constant, we obtain lower and upper bounds on the true probability; the more compilation steps we run, the tighter the probability interval becomes and thus the smaller the approximation error, with the new probability interval included in the previous one. In this sense, this is an anytime approximation algorithm [11].

This lineage compilation approach has been applied to slightly different settings, e.g., top-k query evaluation in the presence of non-materialised views [9], sensitivity analysis and explanation for queries [22], and lifted to lineage of queries with aggregates expressible in the formalism of provenance semimodules [1]. ProApproX evaluates queries on probabilistic XML, where the lineage of the query answer is compiled using the first step only [33]. When this step cannot be applied anymore, ProApproX resorts to Monte Carlo approaches on the subformulas. Top-k queries can be evaluated by incrementally approximating the probabilities of the results using the above anytime algorithm until the lower bounds of k tuples are greater than the upper bounds of the remaining tuples [34].

A further out-database approach that has been coupled with the above compilation approach is model-based approximation [11]: Given a lineage formula Φ, we can derive two formulas Φ_L and Φ_U such that the satisfying assignments of Φ_L are also of Φ, and those of Φ are also of Φ_U. This implies that the probability of Φ_L is less than or equal to that of Φ, which is less than or equal to

[1]The extension to multi-valued variables is straightforward.

that of Φ_U. The benefit of this approach is immediate if the bounds Φ_L and Φ_U admit probability computation in polynomial time and can be derived in polynomial time from Φ.

The first step from the above lineage compilation is also used for computing tractable non-repeating queries with negation [12] and so-called inversion-free unions of conjunctive queries [21]. The processing of such a query critically draws on the observation that the traces of the compilation for subqueries of the input query are OBDDs, whose variable orders are compatible, depths are linear in the database size, and widths only depend on the number of relations in the query. Boolean operations (conjunction, disjunction, negation) on these OBDDs yield OBDDs with the same properties.

In contrast to the lifted approach using inclusion-exclusion that reasons at the query level, the above lineage compilation variants, which rely on grounding the query to its lineage, are limited in that they cannot compute in polynomial time all tractable unions of conjunctive queries [2].

A common out-database probability approximation technique exploits decades-old seminal work on Monte Carlo algorithms including Fully Polynomial-Time Randomized Approximation Schemes (FPRAS) for model counting of propositional formulas in disjunctive normal form [23, 35]. MystiQ and MayBMS use adaptations of such approximations to probabilistic inference: They repeatedly choose at random a possible world and computes the truth value of query lineage. The probability is then approximated by the frequency with which the lineage was true. A common expectation that users have from database management systems is that simple queries run fast and complex queries run slower. Monte Carlo algorithms do not fulfill this expectation because they make no distinction between simple and complex queries. An approach matching the expected behavior would identify simple (tractable) queries and process them at peak performance, while for more complex (hard) queries its performance should degrade smoothly.

HISTORICAL BACKGROUND

The quest for understanding the complexity of probabilistic query processing began two decades ago as a study on query reliability [15], where the Boolean query $R(x), S(x, y), R(y)$ was shown to be hard for #P via a reduction from model counting for positive bipartite formulas in disjunctive normal form. A solid body of work starting a decade later showed complexity dichotomies for various classes of queries.

Query lineage and PC databases draw on c-tables, a formalism for incomplete information put forward three decades ago [19]. It has been formally investigated more recently in the context of provenance semirings [16] and semi-modules [1]. Intensional query semantics was first used in probabilistic databases by Fuhr and Rölleke [13].

MystiQ pioneered in-database techniques for exact query processing on TI databases. SPROUT pioneered in- and out-database techniques based on lin-

eage compilation [11]. Reminiscent of lifted inference in AI, follow-up work lifts compilation to first-order lineage [9].

A detailed account to query processing on uncertain relational and XML data is given in several recent research monographs [34, 18, 4, 24].

KEY APPLICATIONS

Probabilistic databases have a very diverse set of applications [34]. They arise naturally in data integration settings owing to approximate schema and data mappings. Automated information extraction and classification using machine learned models also generate probabilistic facts that can be naturally represented using a probabilistic database. Data cleaning, which involves resolving inconsistencies in data as well as inferring missing values, is another source of uncertainty; likewise, data generated by physical devices that succumb to faults and measurement errors, which is common in scientific databases and sensor networks. In all of these settings, probabilistic databases allow to represent the underlying uncertainties in a unified, consistent way and enabling ad-hoc querying over the uncertain data.

CROSS REFERENCES

Uncertain Data Models; Data Complexity of Query Evaluation;
[outside database work] Model Counting; #P Complexity Class;

Recommended Reading

[1] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.

[2] P. Beame, J. Li, S. Roy, and D. Suciu. Counting of query expressions: Limitations of propositional methods. In *ICDT*, pages 177–188, 2014.

[3] M. Benedikt, E. Kharlamov, D. Olteanu, and P. Senellart. Probabilistic XML via Markov Chains. *PVLDB*, 3:770–781, 2010.

[4] L. Chen and X. Lian. *Query Processing over Uncertain Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[5] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16:523–544, 2007.

[6] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[7] N. N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

[8] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge Vault: A Web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, pages 601–610, 2014.

[9] M. Dylla, I. Miliaraki, and M. Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *ICDE*, pages 122–133, 2013.

[10] R. Fink, A. Hogue, D. Olteanu, and S. Rath. SPROUT^2: A squared query engine for uncertain web data. In *SIGMOD*, pages 1299–1302, 2011.

[11] R. Fink, J. Huang, and D. Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, pages 823–848, 2013.

[12] R. Fink and D. Olteanu. Dichotomies for queries with negation in probabilistic databases. *ACM Trans. Database Syst.*, 41(1):4, 2016.

[13] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database syst. *TOIS*, 15:32–66, 1997.

[14] W. Gatterbauer and D. Suciu. Oblivious bounds on the probability of boolean functions. *TODS*, 39(1):5, 2014.

[15] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[16] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[17] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, pages 1071–1074, 2009.

[18] I. F. Ilyas and M. A. Soliman. *Probabilistic Ranking Techniques in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[19] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31:761–791, 1984.

[20] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. The Monte Carlo Database System: Stochastic analysis close to the data. *TODS*, 36(3):18, 2011.

[21] A. K. Jha and D. Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3):403–440, 2013.

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

[23] R. M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10:429–448, 1989.

[24] B. Kimelfeld and P. Senellart. Probabilistic XML: Models and complexity. In *Advances in Prob. Db. for Uncertain Inf. Mgt.*, pages 39–66. 2013.

[25] X. Lian and L. Chen. Efficient query answering in probabilistic RDF graphs. In *SIGMOD*, pages 157–168, 2011.

[26] D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

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

[28] C. Ré, N. N. Dalvi, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Eng. Bull.*, 29(1):25–31, 2006.

[29] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, pages 715–728, 2008.

[30] C. Ré and D. Suciu. The trichotomy of having queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009.

[31] P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

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

[33] A. Souihli and P. Senellart. Optimizing approximations of DNF query lineage in probabilistic XML. In *ICDE*, pages 721–732, 2013.

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

[35] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[36] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.