Challenges for Efficient Query Evaluation on Structured Probabilistic Data

Antoine Amarilli[1], Silviu Maniu[2], and Mikaël Monet[1]

[1] Télécom ParisTech, Université Paris-Saclay, France
[2] LRI, Université Paris-Sud, Université Paris-Saclay, France

Abstract. Query answering over probabilistic data is an important task but is generally intractable. However, a new approach for this problem has recently been proposed, based on *structural decompositions* of input databases, following, e.g., tree decompositions. This paper presents a vision for a database management system for probabilistic data built following this structural approach. We review our existing and ongoing work on this topic and highlight many theoretical and practical challenges that remain to be addressed.

1 Introduction

To have an accurate description of the real world, it is often necessary to associate probabilities to our observations. For instance, experimental and scientific data may be inherently uncertain, because, e.g., of imperfect sensor precision, harmful interferences, or incorrect modelling. Even when crisp data can be obtained, it can still be the case that we do not trust who retrieved it or how it came to us. The notion of *probabilistic databases* has been introduced to capture this uncertainty, reason over it, and query it: these databases are augmented with probability information to describe how uncertain each data item is. Given a probabilistic database D and a query q, the *probabilistic query evaluation problem* (PQE) asks for the probability that the query q holds on D. Unfortunately, even on the simplest probabilistic database models, PQE is generally intractable [14].

One possibility to work around this intractability is to use approximate approaches, such as Monte Carlo sampling on the data instances. A different direction was recently explored in [2], namely, restricting the kind of *input instances* that we allow, in what we call the *structural approach*. It is shown in [2] that the data complexity of PQE is linear if the instances have *bounded treewidth*, i.e., they can be structurally decomposed in a tree-like structure where each node must contain at most k elements, for a fixed parameter k. Moreover, in [3], it is shown that bounding the instance treewidth is *necessary* to ensure the tractability of PQE, because some queries are hard on *any* unbounded-treewidth instance family (under some conditions). Hence, bounded-treewidth methods seem to be the right way to make PQE tractable by the structural approach.

These theoretical works, however, left open the question of practical applicability: many challenges must still be addressed to implement a practical system using these techniques. First, obtaining an optimal decomposition of an arbitrary instance is NP-hard [5]. Second, the complexity is only polynomial in the *data*, with the query and parameter being fixed; this hides a constant which can be exponential in the width k and non-elementary in the query q. Third, we do not

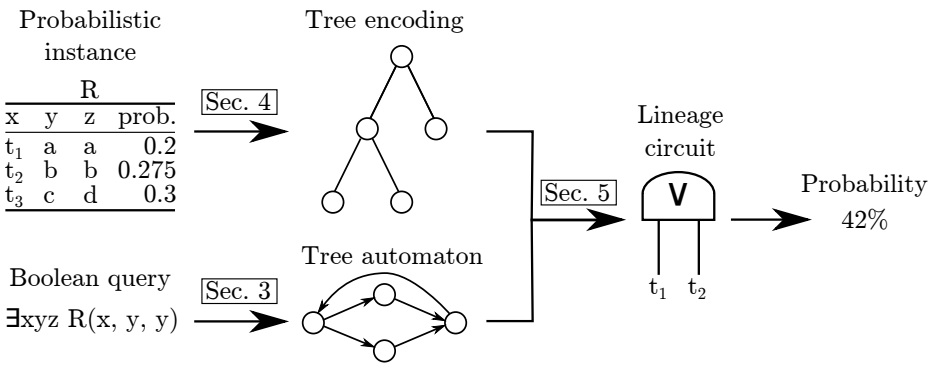

Fig. 1: Overview of the structural approach for PQE

know which real datasets can indeed be decomposed, at least partially, with a small k.

This paper thus presents our vision of a database management system based on the structural approach, and gives an overview of the research directions, both theoretical and practical, which we intend to address to this end.

2 Probabilistic Query Evaluation: A Structural Approach

We first review our *structural* approach [2] for probabilistic query evaluation (PQE). The approach is illustrated in Figure 1.

The approach applies to *tuple-independent (TID) instances* (but generalizes to more expressive models [1]). Formally, a TID instance I is a relational database D where each tuple $t \in D$ has some probability p_t. The TID I represents a probability distribution over the subinstances $D' \subseteq D$ (subsets of facts): following the independence assumption, the probability of D' is $\prod_{t \in D'} p_t \times \prod_{t \in D \setminus D'} (1 - p_t)$.

We study the *probabilistic query evaluation* (PQE) problem: given a *Boolean query* q and TID instance I, determine the probability that q holds on I, i.e., the total probability of the subinstances of I that satisfy q. We refer to the *combined complexity* of PQE when I and q are given as input; we refer to *data complexity* when I is the input and q is fixed.

The first step of the structural approach (Section 3) is to translate the query q to a formalism that can be efficiently evaluated. In the approach of [2], following [13], the query is compiled to a *tree automaton*, i.e., a finite-state automaton over trees [12]. The approach works for expressive queries written in *monadic second-order logic*, which covers in particular first-order logic and (unions of) conjunctive queries. This translation of the query is independent from the instance, so does not affect data complexity; however, it depends on a *parameter* k of the instance, to be defined soon. Intuitively, the automaton represents an algorithm to evaluate the query on suitable instances.

The second step (Section 4) applies to the instance I, and computes a *structural decomposition* of it. In [2], we compute a *tree decomposition* [10, 11], equivalent to junction trees in graphical models [19], and then a *tree encoding* over a finite alphabet: the results of [3] show that tree decompositions are essentially the only

possible way to make PQE tractable. The *parameter* k measures how well I could be decomposed: in our case, k is the *treewidth*, measuring how close I is to a tree. By *treelike* instances, we mean instances whose treewidth is bounded by a constant.

The third step (Section 5) is to compute a *lineage* of the query q on the instance I, i.e., compute an object that represents concisely the subinstances of I that satisfy q. This object can be used for PQE, as what we want to compute is precisely the total probability of this set of subinstances. Specifically, we compute a *Boolean lineage circuit* of the tree automaton for the query over the tree encoding of the instance, according to the construction of [2]. This step is purely symbolic and does not perform any numerical probability computation.

The fourth and last step is to evaluate efficiently the probability of the query from this lineage representation, by computing the probability that the circuit is true. This task cannot be performed efficiently on arbitrary Boolean circuits, but it is feasible in our context, for two independent reasons [2, 3]. First, this circuit can also be tree decomposed, which allows us to apply a message-passing algorithm [19] for efficient probability computation. Second, in the case where we made the query automaton *deterministic* [12], the circuit is actually a d-DNNF [15], for which probabilistic query evaluation is tractable.

3 Efficient Compilation to Expressive Automata

Compiling the query to an automaton following the structural approach of [2], by applying [13], is generally non-elementary in the query. This section presents our main ideas to address this problem: we intend to restrict to *tractable query fragments*, and to use *more expressive automata targets* to compile the query more efficiently. These challenges are not specific to PQE; the next section presents the lineage computation tasks, which are specific to PQE.

Efficient Compilation. Of course, we cannot hope to compile all *Monadic Second Order* logic (MSO) to automata efficiently, or even all *conjunctive queries* (CQs). Indeed, efficient compilation to automata implies that non-probabilistic query evaluation is also tractable in combined complexity on treelike instances; however, CQs are already hard to evaluate in this sense (even on fixed instances). Hence, we can only hope to compile *restricted* query languages efficiently.

Many fragments are known from earlier work to enjoy efficient combined query evaluation. In the database context, for instance, *acyclic CQs* can be evaluated in polynomial combined complexity [23]. This generalizes to first-order logical sentences that can be written with at most k variables, i.e., FO^k [17]. However, it also generalizes to the *guarded fragment* (GF) [4], whose combined complexity is also PTIME, and where better bounds can be derived if we know the instance treewidth [9]. The tractability of GF, however, does not capture other interesting query classes: reachability queries, and more generally *two-way regular path queries* (2RPQs) and variants thereof [6], as well as Monadic Datalog as in [16].

Our first task would thus be to develop an expressive query language that captures GF, 2RPQs, and Monadic Datalog. Ideally this fragment should be

parameterized, i.e., all CQs or all FO queries q could be expressible in the fragment, up to increasing some parameter k_q, with the compilation being PTIME for fixed k_q but intractable in k_q. We would then develop an efficient algorithm to compile such queries to automata that check them on bounded-treewidth instances, for fixed values of the query parameter k_q and of the treewidth. Our ongoing work in this direction investigates very recent extensions of GF with negation and fixpoints [7, 8], for which compilation to automata was studied as a tool for logical satisfiability. We believe that these results, suitably extended and adapted to query evaluation, can yield to bounded-treewidth automaton compilation methods that covers the query classes that we mentioned.

Expressive Automata Targets. The efficient compilation of queries to automata is made easier by allowing more expressive automaton classes as the target language. In [2], we used *bottom-up tree automata*, which process the tree decomposition of the instance from the leaves to the root. Our idea is to move to more expressive representations, namely, *two-way alternating automata* [12]. These automata can navigate through the tree in every direction (including already visited parts), and thus can be more concise. The notion of *alternation* allows automata to change states based on complex Boolean formulae on the neighboring states, which also helps for concision. Indeed, the expressive languages of [7] are compiled to two-way alternating parity automata, which further use a parity acceptance condition on infinite runs, to evaluate fixpoints.

To make automaton compilation more efficient, another idea is to compile queries to automata with a concise implicit representation. In particular, we can use automata with a *structured* state space: the states are tuples of Boolean values, and the transition function can be written concisely for each coordinate of the tuple as a function of the tuples of child states. It may be possible to capture the tractability of query evaluation for 2RPQs via automaton methods, structuring the state space to memorize separately the regular sublanguages of paths between node pairs.

4 Obtaining Tree Decompositions

Estimating Treewidth. As we have mentioned, computing the treewidth of an instance directly is an NP-hard problem. Hence, a practical system using the structural approach must compute tree decompositions more efficiently, even if this limits us to non-optimal decompositions. We intend to experiment with two main kinds of methods to obtain tree decompositions efficiently: *separator-based* algorithms, which recursively divide the instance based on various heuristics; and *elimination ordering* algorithms, where the nodes in the graph are ordered using some measure and eliminated one by one from the graph [10]. To estimate the quality of our decompositions, we can also estimate *lower bounds* on the instance treewidth: for instance via graph degeneracy or average degree [11].

Query-Specific Decompositions. In some cases, knowledge about the query can help us to obtain better tree decompositions of the instance. A trivial situation is when we know that the query is only on a subset of the database relations:

we can then ignore the others when decomposing. More subtly, if we know that specific *joins* are not made by the query, then we may be able to rewrite the instance accordingly, and lower the treewidth. For instance, if no R- and S-atoms share a variable in the query, then the instance $\{R(a,b), S(b,c)\}$ can be rewritten to $\{R(a,b), S(b',c)\}$, which may lower the treewidth by disconnecting elements. We do not understand this process yet in the general case, but we believe that a theory of lineage-preserving instance rewritings for a given query (or query class) can be developed, using the notion of *instance unfoldings* introduced in [3].

5 Tractable Lineage Targets

Once we have compiled the query to an automaton and decomposed the instance to a tree encoding, our goal is to compute a *lineage representation* of the automaton on the encoding, namely, a representation of the subinstances where the query holds, which we will build as a Boolean circuit. We can then use this to perform PQE, by computing the probability of the query as that of the lineage. In so doing, we need to rely on the fact that the lineage is in a class of circuits for which probability can be efficiently computed.

To this end, a first step towards a practical system is to adapt the methods of [2] to the expressive automaton classes that are needed for efficient query compilation. We believe that this is possible, but with a twist: because two-way automata can navigate a tree in every direction, they may go back from where they came, thus resulting in cyclic runs. Therefore, it seems that the natural lineages that we would obtain for alternating two-way automata are *cyclic Boolean circuits*, which we call *Boolean cycluits*. A semantics for such circuits would need to be defined based on the semantics of automaton runs and reachable states: we believe that the evaluation could follow least fixed-point semantics, and be performed in linear time.

Second, we would need to perform efficient probability computation on these cycluits. One first way to address this would be to eliminate cycles and transform them to tractable classes of Boolean circuits (e.g., d-DNNFs), which we believe can be done assuming bounds on the treewidth of the cycluits. Alternatively, we can apply message passing methods directly on the cycluits [19]; or we can try to rewrite the automaton to produce acyclic circuits or even d-DNNFs directly. All these methods would be generally intractable in the query, which is unsurprising: indeed, PQE is often intractable even for languages with tractable combined complexity, and efficient compilation to automata. It would be interesting, however, to identify islands of tractability; and, in intractable cases, to benchmark the previously mentioned approaches and see which ones perform best in practice.

Another important direction for a practical system is to be able to evaluate queries on instances where facts are not independent, i.e., go beyond the TID formalism. For instance, facts could be present or absent according to a complex lineage, like the cc-instances of [1]. In this context, new methods can be efficient. For instance, if the number of probabilistic events is small, performing *Shannon expansions* on some well-chosen events may make large parts of the instance deterministic, making the query easier to evaluate on these parts.

6 Practical Matters

We now review possible approaches and directions to implement and evaluate the structural approach for PQE on real-world datasets.

Results on Treelike Instances. In [21], the structural approach has been compared with one of the existing probabilistic data management systems, namely MayBMS [18]. The instances considered have been randomly generated to have low treewidth ($\leqslant 7$). The results show that an implementation of the structural approach can perform query evaluation faster than the exact methods of MayBMS, in cases where there are many matches and many correlations between them. Indeed, MayBMS does not take advantage of the fact that the instances are treelike. However, in this work, the queries were compiled to automata by hand rather than automatically, and there was no study of practical datasets.

Practical Datasets and Partial Decompositions. A first question is to extend this study to practical datasets, and to investigate whether such datasets have low treewidth, or whether we can use approximate decompositions or reasonably low treewidth. Our preliminary results suggest that some datasets have high treewidth, but others, in particular transportation networks, have treewidth much smaller than their size. For instance, the OpenStreetMaps graph of Paris has over 4 million nodes and 5 million edges, but we estimated its treewidth to be $\leqslant 521$. We do not know yet of a theoretical reason explaining why transportation networks generally exhibit this property.

However, this bound is still too large to be practical. One way to work around this problem is thus to compute a *partial decomposition* [22] of the instance, i.e., a tree decomposition of a part of the instance whose width is at most k, with k fixed. This results in a structure formed of a forest of instances with treewidth $\leqslant k$, called the *tentacles*, that interface with a *core*, i.e., the remaining facts, whose treewidth is too high and that cannot be decomposed. Our preliminary experiments have shown that, for some transportation networks, a partial decomposition for $k = 10$ results in a core instance whose size is about 10% of the original instance.

This decrease in the size of the core, in turn, can potentially have an immediate effect in the processing of queries. Preliminary results [20] have shown that using partial decompositions of fixed treewidth for probabilistic reachability queries, in conjunction with sampling in the core graph, can make query processing up to 5 times faster.

Tentacle Summarization. An important problem when computing probabilities on partial decompositions is the interface between the tentacles and the core, i.e., we must find a way to *summarize* the tentacles in the core when applying sampling to the core. As the tentacles are treelike, we can efficiently compute probabilities and lineages in them: the goal of summarization is to eliminate the tentacles and replace them by *summary* facts that are added to the core. In the case of simple queries, such as reachability queries, the summary facts can have the same semantics as in the original instance, but this does not seem to

generalize to arbitrary queries: it may even be the case that some queries cannot be rewritten to the summary facts while remaining in the same language.

Having summarized the tentacles, we may also answer queries approximately via *sampling*, using the (exact) tentacle summaries added to the core: as the instance is now smaller, this process can be performed faster.

References

1. A. Amarilli. *Leveraging the Structure of Uncertain Data*. PhD thesis, Télécom ParisTech, 2016. 2016-ENST-0021.
2. A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances. In *Proc. ICALP*, volume 9135, 2015.
3. A. Amarilli, P. Bourhis, and P. Senellart. Tractable lineages on treelike instances: Limits and extensions. In *Proc. PODS*, 2016. To appear.
4. H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic*, 27(3), 1998.
5. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2), 1987.
6. P. Barceló Baeza. Querying graph databases. In *Proc. PODS*, 2013.
7. M. Benedikt, P. Bourhis, and M. Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *Proc. LICS*, 2016. To appear.
8. M. Benedikt, B. Ten Cate, T. Colcombet, and M. V. Boom. The complexity of boundedness for guarded logics. In *Proc. LICS*, 2015.
9. D. Berwanger and E. Grädel. Games and model checking for guarded logics. In *Proc. LPAR*, 2001.
10. H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3), 2010.
11. H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations II. Lower bounds. *Information and Computation*, 209(7), 2011.
12. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications, 2007.
13. B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990.
14. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4), 2007.
15. A. Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Applied Non-Class. Log.*, 11(1-2), 2001.
16. G. Gottlob, R. Pichler, and F. Wei. Monadic Datalog over finite structures of bounded treewidth. *TOCL*, 12(1), 2010.
17. E. Grädel and M. Otto. On logics with two variables. *TCS*, 224(1), 1999.
18. J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *Proc. SIGMOD*, 2009.
19. S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *JRSS Ser. B*, 1988.
20. S. Maniu, R. Cheng, and P. Senellart. ProbTree: A query-efficient representation of probabilistic graphs. In *Proc. BUDA*, 2014.
21. M. Monet. Probabilistic evaluation of expressive queries on bounded-treewidth instances. In *Proc. PhD Symposium of SIGMOD/PODS*. ACM, 2016. To appear.
22. F. Wei. TEDI: Efficient shortest path query answering on graphs. In *Proc. SIGMOD*, 2010.
23. M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, 1981.