# Probabilistic Evaluation of Expressive Queries on Bounded-Treewidth Instances

Mikaël Monet
LTCI, CNRS, Télécom ParisTech
Université Paris-Saclay

Supervisor: Pierre Senellart
PhD starting date: November 2015
Expected graduation: November 2018

## ABSTRACT

Though data uncertainty naturally appears in many real-life situations, traditional database theory and systems tend to assume that the data is reliable and complete. The reason is that of complexity and performance: on arbitrary relational database instances annotated with probabilities, performing exact probabilistic query evaluation is hard. However, a criterion on the shape of the database has been shown in recent work to be sufficient and in some sense necessary to the tractability of this task. Databases whose *treewidth* is bounded by a constant $k$ are exactly those that can be tractably queried, with respect to quantitative uncertainty estimation. But this is a *data complexity* result, that does not take into account the cost in terms of the query or of $k$ – in many cases, this cost is too high for real-world applications. The aim of our PhD research is to study in which circumstances the overall complexity of probabilistic query evaluation can become tractable, aiming at both theoretical and practical results.

## 1. PROBLEM

Uncertainty in data can come in various forms. For instance, when information is automatically extracted from arbitrary web pages, uncertainty can be introduced due to the inherent uncertainty of natural language processing or because the source cannot be trusted. In road monitoring systems, uncertainty may come from lack of recent information about traffic [18]: Is the road congested? Is there a diversion? Due to hardware limitations, in the field of experimental sciences, measurement errors also occur in many databases [6]. Querying these databases without considering this uncertainty can lead to incorrect answers.

A natural way to capture database uncertainty is to represent explicitly all possible states of the data (called the *possible worlds*) and associate to each world a probability value. The probability that a tuple belongs in the answer to a query is then the sum of the probabilities of the possible worlds for which this tuple is in the answer. The problem with this approach is that there can be exponentially many such possible worlds, so that it is not feasible in practice to represent the data and query it in this way. Nevertheless, we can efficiently *represent* uncertain data if we make some independence assumptions: each tuple has a probability to be present or absent independently of the other tuples. This is the *tuple-independent* (or *TID* [21, 13]) framework. More elaborate probabilistic representation systems [7, 24, 17, 19, 27] also exist.

Now that we can concisely represent probabilistic databases, can we efficiently query it? Unfortunately, in general, the answer is no. For instance, consider the following simple conjunctive query $q_{\text{hard}} : \exists xy\ R(x) \wedge S(x, y) \wedge T(y)$. It is #P-hard to compute its probability on arbitrary TID instances [13]. Recall that #P is the complexity class of counting problems whose answer can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine; a typical #P-complete problem is #SAT, that of counting the number of satisfying assignments of a propositional formula [29].

To address the intractability of query evaluation on probabilistic databases, three general approaches have been proposed:

- **Approximate probability computation.** It is always possible to resort to Monte-Carlo sampling [15, 23, 20], which provides an *additive approximation* of the query probability. This is of limited use, however, when probabilities are very low: the running time of Monte-Carlo sampling is quadratic in the desired precision, so we cannot use it in cases where we want a high precision.
- **Restricting the queries.** It is possible to restrict to queries for which probabilistic query evaluation is in PTIME. In fact, we already know a complete characterization [14] of the unions of conjunctive queries whose probability is tractable to evaluate on TID instances. However, as it turns out, many simple queries are already hard to evaluate, such as $q_{\text{hard}}$ above.
- **Restricting the shape of the instances.** Recent work from our group [3] has shown that, when the TID instances are taken to be of *bounded treewidth*,


*SIGMOD'16 PhD Symp, June 25-July 01 2016, San Francisco, CA, USA*

DOI: http://dx.doi.org/10.1145/2926693.2929905

then we can evaluate in linear time (up to the cost of arithmetic operations) the probability of any **fixed** Monadic Second Order (MSO) query $q$. MSO is an extension of first-order logic where quantification over sets of elements is allowed. Notice that this is a *fixed-parameter data complexity* result: both the size $|q|$ of the query $q$ and the upper bound $k$ on the treewidth of the instances are considered to be constant. The resulting algorithm may thus involve a constant factor that is arbitrarily high in $|q|$ and $k$.

Our PhD research aims at combining the best of these three approaches in order to obtain tractable combined complexity of a large class of queries on a large class of data, or in more practical terms to provide the machinery and systems to support real-life applications of probabilistic query evaluation on real-world, large-scale, datasets.

Section 2 discusses in more detail the state of the art in probabilistic databases, and, specifically, the known results about restricting to bounded-treewidth instances. We then present in Section 3 some preliminary results about implementing the theoretical tools from [3] to obtain more efficient probabilistic query evaluation, demonstrating the potential of the approach. This leads us to Section 4 where we present our methodology to go beyond the state of the art.

# 2. STATE OF THE ART

Here we first discuss the representation of probabilistic databases, then we describe the techniques for query evaluation on bounded-treewidth instances.

## 2.1 Probabilistic Databases

As previously discussed, it is not a good idea to represent uncertain relational databases by explicitly describing all of their possible states. One general way to represent them is to use the formalism of *pc-instances* [17]. A pc-instance is a normal relational instance where tuples are annotated by propositional formulas over a set $X$ of Boolean events. A valuation of those events then defines a unique possible world consisting of all tuples whose annotations evaluate to true. By giving independent probabilities to the events in $X$, we define the probabilistic distribution of the possible worlds of our data.

A simpler formalism is that of *tuple-independent instances* (TID [21, 13]), that we mentioned in the introduction. A TID instance is like a pc-instance but where each tuple is directly annotated by the probability that the tuple is present, independently of the other tuples. Of course the TID formalism is less expressive than pc-instances, because there are some probability distributions that can be represented by the latter but not by the former. For instance a TID instance cannot express that two tuples are mutually exclusive. However, probabilistic query evaluation is already hard on TID instances. As this implies that query evaluation is already challenging on that formalism, we will focus on it for the time being, but note that the techniques on [3] extend to pc-instances, given appropriate definitions for their treewidth [2, Section 4.3.1].

An example of TID instance is given in Table 1, describing who likes which dessert. A possible world of this instance would be that Mary likes apple cake, Tom likes meringue and the other facts are absent. This possible world has probability $(1-0.9)\times(1-0.2)\times 0.5\times 0.6 = 0.024$. An interesting query

**Table 1: Dessert preferences**

| Name | Likes | Prob. |
|---|---|---|
| Bob | tiramisu | 0.9 |
| Mary | tiramisu | 0.2 |
| Mary | apple cake | 0.5 |
| Tom | meringue | 0.6 |

on such an instance would be: what is the probability that there exist two different people liking the same dessert? This can be expressed as the probability of the conjunctive query with disequalities $\exists p_1 p_2 d\, \mathrm{Likes}(p_1, d) \wedge \mathrm{Likes}(p_2, d) \wedge p_1 \neq p_2$. The answer is the sum of the probabilities of the possible worlds in which the query is true, which can easily seen to be $0.9 \times 0.2 = 0.18$.

A number of probabilistic database management systems have been developed [1, 20, 19, 26]. In our experiments, we will compare to MayBMS [19], a readily available system[1] that is still maintained, and supports both TIDs and arbitrary pc-tables. MayBMS is implemented as an extension of PostgreSQL and provides techniques for exact and approximate probabilistic inference.

Which queries are such that exact probability computation is tractable on arbitrary TID instances? For unions of conjunctive queries (UCQs), a dichotomy result is provided by the work of Dalvi and Suciu [14]: the data complexity of a given UCQ is either #P-hard (the query is then said *unsafe*, like $q_{\mathrm{hard}}$) or it is in PTIME (the query is *safe*).

## 2.2 Restricting the Treewidth

We now present related work on the efficient evaluation of expressive queries on *treelike* instances. Treelike instances are instances with low *treewidth*, where treewidth is a measure used to quantify how far a graph is from being a tree. For example a tree has treewidth 1, a cycle has treewidth 2, and a $k$-clique has treewidth $k-1$. One way of defining treewidth is by using the notion of *tree decomposition* of a graph. A tree decomposition is a tree labeled by sets of vertices of the graph with some additional properties being satisfied. Formally in the case of graphs, letting $2^V$ be the powerset of $V$:

Definition 1. *Let $G = (V, E)$ be a finite graph with vertex set $V$ and edge set $E$. A* tree-decomposition *$Tdec = (T, \lambda)$ of $G$ is a tree $T$ with labeling function $\lambda : T \to 2^V$ such that:*

1. *For every edge $(a, b) \in E$ there exists a node $n \in T$ for which $\{a, b\} \subseteq \lambda(n)$.*
2. *For every vertex $v \in V$, the set $T_v = \{n \in T \mid v \in \lambda(n)\}$ is a subtree of $T$.*

*The* width *of a tree-decomposition is the size of its largest bag $w(Tdec) = \max_{n \in T} |\lambda(n)| - 1$. Finally, the* treewidth *of $G$ is the minimal width of a tree-decomposition of $G$.*

Treewidth generalizes quite easily to any relational instance: the treewidth of a relational instance $I$ is that of its *primal graph*, whose nodes are the elements of the domain of $I$ and where there is an edge between elements $a$ and $b$ if and only if there is a fact $R(\overline{x})$ of $I$ in which $a$ and $b$ co-occur $(a, b \in \overline{x})$.

[1] See http://maybms.sourceforge.net/.

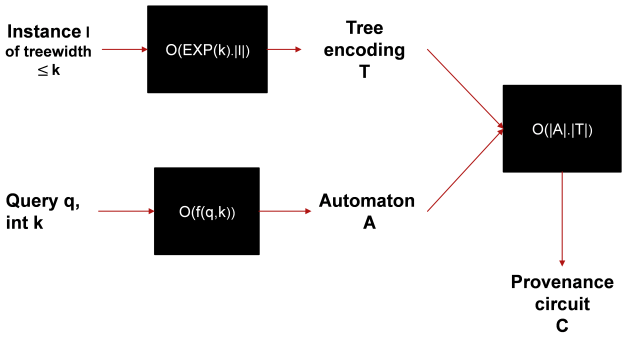


**Figure 1: Obtaining the provenance circuit**

While the definition of treewidth may seem complex, it is a well-known natural criterion to ensure the tractability of many problems that are NP-hard on arbitrary instances. An intuitive way to understand tree-decompositions is that they provide a way to decompose the instance in such a way as to be able to use *divide and conquer* algorithms.

Courcelle showed in 1990 [12, 16] that evaluating an arbitrary MSO query on bounded-treewidth instances can be done in time linear in the instance. To do so, the query $q$ is compiled into a *tree automaton* $A$, independently from the data but depending on the treewidth parameter $k$. The instance $I$ of treewidth $\leq k$ is then transformed up to isomorphism into what we call a *tree-encoding* $T$, which is just a tree-decomposition encoded with a finite alphabet so as to be processable by a tree automaton. Then $T$ is accepted by $A$ iff $q$ holds on $I$ (written $I \models q$).

From this, [3] showed that we can compute, always in linear time in the instance, a circuit $C$ that contains more information on why the query is true or false: the *provenance circuit* of the query $q$ on the instance $I$.

Definition 2. *The* provenance circuit $C_{q,I}$ *of a query* $q$ *on instance* $I$ *is a boolean circuit whose input gates are the facts of* $I$ *and such that for every valuation* $\nu : I \to \{0,1\}$*,* $C_{q,I}$ *evaluates to* 1 *under* $\nu$ *if and only if* $\nu(I) \models q$*, where* $\nu(I)$ *consists in the subinstance* $\{\bar{t} \in I \mid \nu(\bar{t}) = 1\}$*.*

Thus, the provenance circuit allows us to know the output of the query on any subinstance of $I$. Thanks to this ability and because the computed circuit is itself treelike, it can be used to compute the probability that $q$ holds on $I$, still in linear time (modulo the cost of arithmetic operations). The global picture can be seen in Figure 1.

There are two major drawbacks to this method:

- We need to know the treewidth $k$ of the instance in order to transform it into a tree-encoding. Sadly, determining the treewidth of an arbitrary instance is an NP-hard problem [5]. However the treewidth can be approximated by lower- and upper-bound heuristics that usually give adequate results, especially when the treewidth is low [8, 9, 30].
- The data complexity may be linear, but the complexity of computing the automaton is non-elementary in the query [28] and exponential in the treewidth [12].

As a followup to [3] and in a very recent work [4], it has been shown that bounded-treewidth is essentially the only relevant notion for instance-based tractability of probabilistic query evaluation: if a family of instances does not have bounded treewidth, then there exists a query (actually independent of the family) such that probabilistic evaluation on this family is intractable (this has been shown in arity two and when such families allow efficient construction of instances).

# 3. PRELIMINARY RESULTS

Our first direction was to investigate whether the automaton-based techniques relying on bounded-treewidth instances from [3] have potential for application, despite their drawbacks.

We have implemented a framework using these techniques, and have compared it with MayBMS on randomly generated probabilistic graphs of low treewidth for 4 different queries. To generate graphs of low treewidth, we generate trees whose edges are annotated by a relation name $\in \{R, S, T\}$ and a probability. This gives us a database on a signature containing only three binary relation symbols, with treewidth 1. To increase the treewidth we add a few more edges between some nodes that are not already connected.

We rely on an external library [30], which, from a graph, can compute its treewidth and a tree-decomposition, or at least (using heuristics) an upper-bound on the treewidth and a tree-decomposition of non-minimal width. The timings to get the tree-decompositions are decent when the treewidth is low. From the tree-decompositions we compute the tree-encodings in linear time. We point out that the tree-encoding is not specific to the query, so the step needs to be performed only once per instance.

We used another library [10] to represent tree automata. We have not implemented an algorithm to compute the automaton from the query, and we have so far compiled queries to automata by hand. We optimized the algorithms of [3] to make them faster and to produce smaller provenance circuits. One of the optimizations is to compute automaton states, not before building the provenance circuit, but at the same time it is built. That way we compute only the rules of the automaton that are needed by the instance, because the whole automaton can be huge.

The first three queries that we compiled can be expressed as simple SQL `SELECT-FROM-WHERE` queries and we have benchmarked our method by computing their probabilities on the generated instances. We use MayBMS as a baseline implementation against which we compare the performance of our method. We take one of these queries, $q_3$, as representative of the results. On Figure 2 we compare the time needed to evaluate the probability of $q_3$ as a function of the number of facts of some instances by our implementation with that of MayBMS (approximate computations only, the exact computations all resulted on a timeout on this query). The instances all have treewidth between 2 and 7. The timing for our method includes obtaining the tree-encoding, building the circuit and the automaton, and computing the probability. Because the instances do not all have the same treewidth, there are irregularities in the graph (the time needed to obtain the tree-encoding is highly dependent on the treewidth). In MayBMS, the approximate computation operator aconf$(x, y)$ gives the value at $\pm x$ with probability $\geq (1 - y)$. We set a timer to stop the computations at 30s, the missing values in Figure 2 are cases in which these 30s were exceeded.

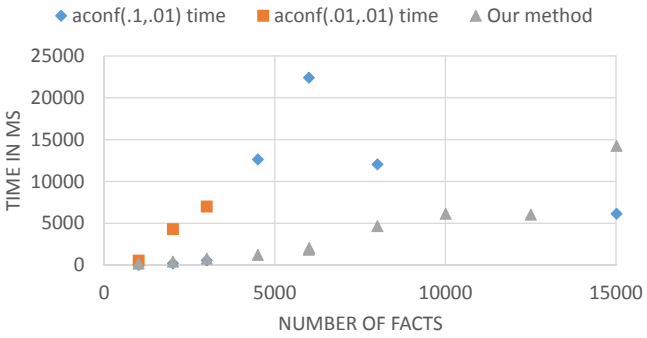


**Figure 2: Timing comparison on query $q_3$ between MayBMS approximate computation and our tree-decomposition technique**

The query is

$$q_3 \equiv \exists x \exists y, x \neq y \wedge (\text{isInR}(x) \wedge \text{isInS}(x)) \\ \wedge (\text{isInR}(y) \wedge \text{isInS}(y))$$

where $\text{isInR}(x) \equiv \exists z, R(x,z) \vee R(z,x)$ and similarly for IsInS.

Such a query asks if there exist two different elements that are concerned by the relations $R$ and $S$. We see that our method obtains better results in almost all cases. This happens because in this case there are a lot of matches and many correlations between these matches. We take advantage of the data being treelike, while MayBMS has to go through some exponential steps to factorize the numerous correlations.

The fourth query, which expresses connectivity, cannot be handled by MayBMS because MayBMS only supports first-order queries, and is reasonably efficiently executed by our approach. We must add that MayBMS is written in C and is quite optimized while we coded in Java and with less care of details.

These preliminary experiments suggest that there are indeed cases when query evaluation on probabilistic databases can be made faster by decomposing instances. However, two important limitations remain:

- in general, we do not have an efficient way to obtain the tree automaton for the query;
- the approach is only applicable to low-treewidth data instances.

We now report on work in progress that aims at overcoming these limitations.

# 4. WORK IN PROGRESS

We present in this section two of the possible directions that we intend to follow during our PhD, both of which aim at showing that the treewidth approach can be used for realistic applications.

## 4.1 Lowering the Combined Complexity

The goal here would be to overcome the non-elementary complexity in the query to build the automaton. Indeed if we want to automatically compute the automaton, such a complexity seems to be a problem for efficiency, even if the query is small. Our general idea to work around this issue is to compile queries to more expressive automata formalisms. Indeed, in the standard presentation of Courcelle's results, the automata used are *bottom-up automata*, that is, automata which process the tree-encoding from the leaves to the root. We are currently investigating how much we could gain by using different kinds of automata, for example tree automata that can go in every direction, namely *two-way tree automata* [11].

Indeed it is known that some regular languages on words are recognized by deterministic automata that can be exponentially bigger if they read the word in the wrong direction. Consider for instance the language on the alphabet $\{a, b\}$ of the words whose $n$-th symbol from the end is a $b$; if the automaton reads the word from left to right then we need roughly $2^n$ states [25], but if it reads it from right to left then we only need $n$ states. We are currently trying to determine which query languages can be tractably compiled into automata, with the hope of achieving polynomial or even linear complexity in the query. Our goal is still to create provenance circuit for treelike instances, by a generalization of the bounded-treewidth methods to more expressive tree automata classes. Our work appears to lead to provenance circuits with *cycles*, which to our knowledge would be a novelty in the world of provenance.

## 4.2 Real-World Applications

Thanks to our implementation, we know that the treewidth approach is suitable for databases that are of low treewidth. Therefore, the first step towards the implementation of a real-life application would be to identify which real databases meet this criterion. We therefore aim at computing the treewidth of various kinds of networks, such as transportation networks, large genealogical trees (which are actually not trees), graphs from social networks, etc. We suspect that transportation networks of star-shaped cities are good candidates. The heuristics to obtain the tree-encoding have to be tested on these various databases.

Moreover, we are also investigating two new ideas to make bounded-treewidth methods more widely applicable, the first being to tree-decompose the instance *for a given query*. Imagine for example that a query does not mention a relation $R$, then we can remove $R$ before tree-decomposing the instance, which may make the treewidth lower. More generally, we hope to rely on the fact that for some particular classes of UCQs, in particular the *inversion-free* UCQs of [14], it was shown in [4] that any instance can be rewritten to a bounded-treewidth instance without modifying the provenance circuit. This connects the safe queries approach to the treewidth-based one, in the sense that such a safe query "sees" TID instances as being of bounded treewidth. We intend to generalize this approach and characterize, for each specific query, the instances that can be equivalently rewritten (in a lineage-preserving way) to bounded-treewidth instances.

The second idea is to combine the treewidth and automata approach with approximation methods. For an instance that is not easily tree-decomposable, we would try to tree-encode only the parts that are treelike (that we call the *tentacles*) and then combine this with some approximation techniques on what remains (the *core*). This idea was applied in [22] to a simpler decomposition technique (SPQR-trees) and for a specific family of queries (source-to-target queries in a graph). One possible technique would be to rewrite the query on the core to be able to use the provenance circuits that come

from the tentacles. However, this seems challenging: when using the approximation techniques on the core, we would need to sample efficiently what is needed. For instance, in the case of a reachability query, rewriting the query to the core can be tricky given that a query match may go from one tentacle to another while passing through the core. Also, can we notice cases where the query will often by satisfied by the core alone, or by a tentacle alone, and optimize for such cases?

Furthermore, we can combine these ideas with those of Section 4.1, so that the automata that will handle the treelike tentacles can be built as efficiently as possible.

# 5. CONCLUSION

We gave an overview of the world of probabilistic databases, and of the different approaches that have been proposed to lower the complexity of probabilistic query evaluation: approximate the probabilities, restrict the class of queries, and restrict the shape of instances. So far, we have shown that the treewidth and automata techniques can be practically suitable for exact probability computation in cases where the data has low treewidth, which was not a given.

We then described the intended direction of our PhD work: combining the best of the three approaches to obtain theoretical and practical results in view of realistic applications.

# ACKNOWLEDGMENTS

This material is based on work supported by the Télécom ParisTech Research Chair on Big Data & Market Insights.

# 6. REFERENCES

[1] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.

[2] A. Amarilli. *Leveraging the Structure of Uncertain Data*. PhD thesis, 2016.

[3] A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances. In *Proc. ICALP*, 2015.

[4] A. Amarilli, P. Bourhis, and P. Senellart. Tractable lineages on treelike instances: Limits and extensions. In *Proc. PODS*, 2016.

[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] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting protein complex membership using probabilistic network reliability. *Genome Research*, 14(6), 2004.

[7] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *TKDE*, 4(5), 1992.

[8] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3), 2010.

[9] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations II. Lower bounds. *Inf. Comput.*, 209(7), 2011.

[10] P. Claves, D. Jansen, S. J. Holtrup, M. Mohr, A. Reis, M. Schatz, and I. Thesing. LETHAL, 2009. Available from http://lethal.sourceforge.net/.

[11] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications, 2007. Available from http://www.grappa.univ-lille3.fr/tata.

[12] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990.

[13] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4), 2007.

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

[15] G. S. Fishman. A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness. *IEEE Trans. Reliability*, 35(2), 1986.

[16] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.

[17] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *Proc. IIDB*, 2006.

[18] M. Hua and J. Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, 2010.

[19] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *Proc. SIGMOD*, 2009.

[20] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[21] L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. ProbView: A flexible probabilistic database system. *TODS*, 22(3), 1997.

[22] S. Maniu, R. Cheng, and P. Senellart. ProbTree: A query-efficient representation of probabilistic graphs. In *Proc. BUDA*, 2014.

[23] C. Ré, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. ICDE*, 2007.

[24] C. Ré and D. Suciu. Materialized views in probabilistic databases: For information exchange and query optimization. In *Proc. VLDB*, 2007.

[25] A. L. Rosenberg. *The Pillars of Computation Theory: State, Encoding, Nondeterminism*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[26] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. The orion uncertain data management system. In *Proc. COMAD*, 2008.

[27] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[28] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2(1), 1968.

[29] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3), 1979.

[30] T. van Dijk, J.-P. van den Heuvel, and W. Slob. Computing treewidth with LibTW. Master's thesis, University of Utrecht, 2006. Available from http://treewidth.com/treewidth/docs/LibTW.pdf, software available from https://github.com/WPettersson/libtw.