Author manuscript, published in "EDBT Workshop: The Logic in Databases (2011)"

inria-00591905, version 1 - 10 May 2011

Value Joins are Expensive over (Probabilistic) XML*

Evgeny Kharlamov
Free University of Bozen-Bolzano
Piazza Domenicani 3
39100 Bolzano, Italy
kharlamov@inf.unibz.it

Werner Nutt
Free University of Bozen-Bolzano
Piazza Domenicani 3
39100 Bolzano, Italy
nutt@inf.unibz.it

Pierre Senellart
Institut Télécom; Télécom ParisTech
CNRS LTCI, 46 rue Barrault
75634 Paris, France
pierre.senellart@telecom-paristech.fr

ABSTRACT

We address the cost of adding value joins to tree-pattern queries and monadic second-order queries over trees in terms of the tractability of query evaluation over two data models: XML and probabilistic XML. Our results show that the data complexity rises from linear, for join-free queries, to intractable, for queries with value joins, while combined complexity remains essentially the same. For tree-pattern queries with joins (TPJ) the complexity jump is only on probabilistic XML, while for monadic second-order logic over trees with joins (TMSOJ) it already appears for deterministic XML documents. Moreover, for TPJ queries that have a single join, we show a dichotomy: every query is either essentially join-free, and in this case it is tractable over probabilistic XML, or it is intractable. In this light we study the problem of deciding whether a query with joins is essentially join-free. For TMSOJ we prove that this problem is undecidable and for TPJ it is Π_2^{P}-complete. Finally, for TPJ we provide a conceptually simple criterion to check whether a given query is essentially join free.

1. INTRODUCTION

Uncertainty is ubiquitous in data and many applications must cope with this: information extraction from the World Wide Web [7] or automatic schema matching in information integration [23] are inherently imprecise. This uncertainty is sometimes represented as the *probability* that the data is correct, as with conditional random fields [19] in information extraction, or uncertain schema mappings in information integration [12]. In other cases, only *confidence* in the information is provided by the system, which can be seen after renormalization as an approximation of the probability. It makes sense to manipulate this probabilistic information in a *probabilistic database management system* [9].

Recent work has proposed models for probabilistic data, both in the relational [29, 10, 18] and XML [22, 2, 17] settings. We focus here on the latter, which is particularly adapted in the case, common on the Web, when the information is not strictly constrained by a schema, or when it is inherently tree-like (mailing lists, parse trees of natural language sentences, etc.). A number of works on probabilistic XML have dealt with query answering for a variety of models and query languages [22, 17, 1, 4]. On the other hand, queries with value joins, equating character data in different fragments of XML documents, have received far less attention, with the exception of [1]. This is despite the fact that value joins proved their importance in SQL: joins are at the core of SELECT-PROJECT-JOIN, the most used fragment of SQL. We propose in this article a general study of the complexity of query answering in both XML and probabilistic XML with hierarchical probabilistic dependencies [27, 2] (see Section 2 for details).

The first work addressing join queries for probabilistic XML data is [1] where we showed that adding joins to tree-pattern queries (TP) significantly increases complexity of query answering. This is a *data complexity* [28] result, i.e, we measured the complexity in the size of the input probabilistic XML data, and the query is assumed to be fixed, in contrast to *combined complexity*, where both the data and the query are parts of the input.

The complexity shift was shown by exhibiting a #P-hard query (see Section 3 for details), which shows that the whole class of tree-pattern queries with joins (TPJ), is intractable. Recall that for TP, and, indeed, for all of monadic second-order logic over trees (TMSO), which subsumes TP, query evaluation is linear in data complexity. This raises a number of questions about query language with joins, that this work aims at answering:

- What is the precise complexity of query evaluation, over XML and probabilistic XML, for TPJ and TMSO extended with joins (TMSOJ), both in data and combined complexity?
- We say that a TMSOJ (resp., TPJ) query has an *essential join* if it is *not equivalent* to a TMSO (resp., TP query). A query without essential join is called an *essentially join-free* query. Is it possible to decide, given a TMSOJ or TPJ query, whether this query really has any essential join?
- Is the fact that a query has an essential join responsible for making it hard? In other words, are all queries with essential joins hard to evaluate?

The rest of the paper is devoted to answering these questions. In Section 2 we introduce deterministic and probabilistic data models, tree-pattern and monadic second-order queries, and review related work on query answering for these models. In Section 3 we extend the query models with joins. In Sections 4 and 5 we study the complexity of joins over XML and probabilistic XML as well as the complexity of deciding essential joins. **Due to lack of space we omit some proofs that can be found in a technical report [16].**

2. PRELIMINARIES

We briefly define in this section the data model (see, e.g., [2] for more details) and the query languages we use.

*This work has been partially funded by ERC FP7 grant Webdam (agreement 226513).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
LID 2011 March 25, 2011, Uppsala, Sweden
Copyright 2011 ACM 978-1-4503-0609-6 ...$10.00.

inria-00591905, version 1 - 10 May 2011

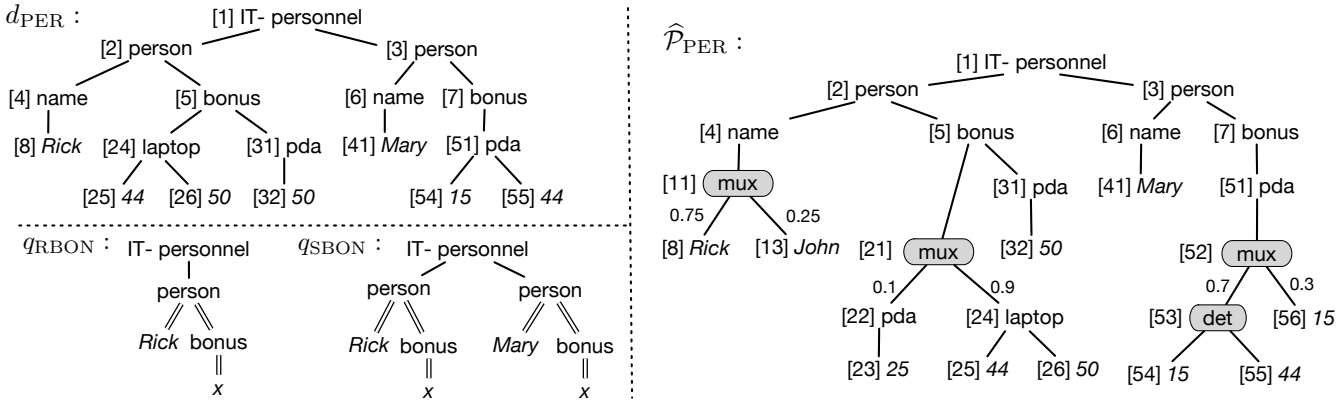


Figure 1: Example p-document $\widehat{\mathcal{P}}_{\mathrm{PER}}$, possible document d_{PER} of $\widehat{\mathcal{P}}_{\mathrm{PER}}$, TP query q_{RBON} and TPJ query q_{SBON}

Documents. We assume a countable set of labels $\mathcal{L}$. We model an XML *document* d as an unranked, labeled, and unordered tree. Results of this paper can be extended to ordered trees. We say that two documents d_1 and d_2 are *equivalent*, denoted $d_1 \sim d_2$, if they share the same structure and labels, i.e., if there is a bijection between the nodes of d_1 and d_2 preserving the edges, root, and labels.

EXAMPLE 1. *Consider the document d_{PER} in Figure 1 (top-left) describing the personnel of an IT department and the bonuses for different projects. It indicates that* Rick *worked under two projects (*laptop *and* pda*) and got bonuses of 44 and 50 in the former project and 50 in the latter one. Identifiers are in brackets before labels.*

We define a *finite probability space of XML documents*, or *px-space* for short, as a pair $(\mathcal{D}, \Pr)$ with $\mathcal{D}$ a set of pairwise non-equivalent documents and $\Pr$ mapping every document d to a probability $\Pr(d)$ such that $\sum\{\Pr(d) \mid d \in \mathcal{D}\} = 1$.

Probabilistic documents. Following [2], *p-documents* are a general syntax for compactly representing px-spaces. Like a document, a p-document is a tree but is has two kinds of nodes: *ordinary* nodes, which have labels and are the same as in documents, and *distributional* nodes, which are used to define the probabilistic process for generating random documents. We consider in this work two kinds of distributional nodes, namely, *mux* (for *mutually exclusive*) and *det* (for *deterministic*). Other kinds of distributional nodes are studied in [2], but as shown there, *mux* and *det* alone are enough to represent all px-spaces as p-documents, and *mux-det* p-documents can be seen as XML counterparts of the block-independent databases of Dalvi, Ré, and Suciu [9]. An important characteristic of these distributional nodes that will play a fundamental role in the tractability of query evaluation is the fact that they only describe *local* probabilistic dependencies in the tree.

Formally, a p-document $\widehat{\mathcal{P}}$ is an unranked, unordered tree, with labels in $\mathcal{L} \cup \{\mathit{mux}(\Pr)\} \cup \{\mathit{det}\}$. If a node v is labeled with $\mathit{mux}(\Pr_v)$ then $\Pr_v$ assigns to each child v' of v a probability $\Pr_v(v')$ with $\sum_{v'} \Pr_v(v') \leq 1$. We require the leaves and the root of a p-document to be ordinary nodes, that is, with labels in $\mathcal{L}$. The class of all *mux-det* p-documents is denoted $\mathsf{PrXML}^{\mathit{mux,det}}$.

EXAMPLE 2. *Figure 1 (right) shows a p-document $\widehat{\mathcal{P}}_{\mathrm{PER}}$ that has mux and det distributional nodes, shown on gray background. Node n_{52} is a mux node with two children n_{53} and n_{56}, where $\mathsf{Pr}_{n_{52}}(n_{53}) = 0.7$ and $\mathsf{Pr}_{n_{52}}(n_{56}) = 0.3$.*

A p-document $\widehat{\mathcal{P}}$ has as associated *semantics* a px-space $[\![\widehat{\mathcal{P}}]\!]$ defined by the following random process. Independently for each $\mathit{mux}(\Pr_v)$ node, we select at most one of its children v' and delete all other children with their descendants. We do not delete any of the children of *det* nodes. We then remove in turn each distributional node, connecting its ordinary children with their lowest ordinary ancestors. The result of this process is a random document $\mathcal{P}$, and the probability of a specific run $p_{\mathcal{P}}$ is the product of all $\Pr_v(v')$ for each chosen child v' of a *mux* node v; when we choose no children for a *mux* node v, we multiply by $1 - \sum_{v'} \Pr_v(v')$ instead. Since there may be several different ways to generate a document equivalent to $\mathcal{P}$, we define the probability $\mathsf{Pr}(\mathcal{P})$ of a random document $\mathcal{P}$ as: $\mathsf{Pr}(\mathcal{P}) := \sum_{d \sim \mathcal{P}} p_d$. It is easy to see that if we select one representative of each equivalence class, we obtain a px-space.

EXAMPLE 3. *The only way to obtain a document equivalent to d_{PER} at Figure 1 from $\widehat{\mathcal{P}}_{\mathrm{PER}}$ is to choose: the left child of n_{11}, the right child of n_{21}, and the left one of n_{52}. The probability of these choices, and the probability of d_{PER}, is $0.4725 = 0.75 \times 0.9 \times 0.7$.*

Queries over documents. We now define two Boolean query languages over XML documents, namely monadic second-order queries and tree-pattern queries. We first define join-free versions of these query languages and add value joins in Section 3. A Boolean query over documents can be seen as a mapping from every XML document to either true or false. We thus say that a query q *is true in a document* d, or that d *is a model of* q, and we note $d \models q$. Given two queries q and q', we say that q is contained in q', denoted $q \sqsubseteq q'$, if all models of q are also models of q'. Two Boolean queries are *equivalent*, denoted $q \equiv q'$, if $q \sqsubseteq q'$ and $q' \sqsubseteq q$.

Monadic second-order queries. The first query language we consider is *monadic second-order logic over trees* (TMSO for short), a general query language that has the property of having linear data complexity, due to the fact that every TMSO query can be converted (in non-elementary time) into a deterministic bottom-up tree automaton [25]. TMSO is more expressive than other classical join-free XML query languages such as tree-pattern queries with Boolean operators [17] or navigational XPath [5].

TMSO is the logic built up from: (i) unary predicates for labels: for every $l \in \mathcal{L}$ there is a unary predicate $\mathsf{Label}_l(\cdot)$; (ii) the binary *child* relation $\mathsf{Ch}(\cdot,\cdot)$; (iii) node variables; and (iv) monadic (i.e., unary) predicate variables via Boolean connectors, and first and second-order quantifiers $\exists x, \exists S$. Since we consider only Boolean queries, we assume all variables occurring in a TMSO query are bound by a quantifier. The semantics of TMSO is standard [25]. A *descendant* predicate Desc can be expressed in TMSO using the Ch predicate and quantifiers; we will use Desc when needed as if it were part of TMSO. The language TFO of first-order queries on trees is TMSO without any second-order quantifiers.

inria-00591905, version 1 - 10 May 2011

Tree-pattern queries. Monadic second-order logic is a very expressive query language on trees. The language of *tree-pattern queries* (TP), roughly the subset of navigational XPath with only *child* and *descendant* axes, has more limited expressive power, but, as we shall discuss, more efficient query evaluation algorithms. Let $\mathcal{X}$ be a countable set of variables, disjoint from $\mathcal{L}$. A tree-pattern query is an unordered, unranked tree, with labels in $\mathcal{L} \cup \mathcal{X}$, where edges are labeled either with *child*, or *descendant* types. A variable from $\mathcal{X}$ may not be used twice in the same query (we will obviously remove this assumption when we introduce joins). Note that in TMSO variables denote nodes, while in TP, with a slight overloading, they denote (unknown) labels of nodes. A *subquery* of $q \in$ TP is any subtree of q. A TP query q is true in a document d if and only if there is a mapping ν from the nodes of the query to the nodes of d such that: (i) if r is the root of q, then $\nu(r)$ is the root of d; (ii) if u, v are two nodes of q connected by a child edge, $\nu(v)$ is a child of $\nu(u)$; (iii) if u, v are two nodes of q connected by a descendant edge, $\nu(v)$ is a descendant of $\nu(u)$; (iv) unless u is labeled with a variable of $\mathcal{X}$, u and $\nu(u)$ have the same labels. A mapping satisfying all these conditions is also called a *homomorphism*.

EXAMPLE 4. *Consider the query q_{RBON} in Figure 1 (bottom-right) asking whether* Rick *has received any bonus, i.e., a bonus x. Single lines denote child edges and double lines descendant edges.*

Querying p-documents. Up to now, we have seen Boolean queries as Boolean functions over documents. Over a p-document, a Boolean query naturally yields a probability: the probability this query is true in the set of possible documents defined by this document. More formally, given a query q and a p-document $\widehat{\mathcal{P}}$, the semantics of q over $\widehat{\mathcal{P}}$ is the probability $q(\widehat{\mathcal{P}}) := \sum_{\substack{d \in [\![\widehat{\mathcal{P}}]\!] \\ d \models q}} \Pr(d)$.

This definition yields an algorithm for computing the probability of a Boolean query over a p-document, given an algorithm for determining whether a query is true in a document: just enumerate all possible worlds, evaluate the query over each of these, and sum up the probability of documents satisfying the query. This algorithm is exponential-time, however, and it is often possible to be more efficient than that.

EXAMPLE 5. *The query q_{RBON} is true in d_{PER} since Rick indeed received bonuses. Evaluation of q_{RBON} over $\widehat{\mathcal{P}}_{\text{PER}}$ returns true if and only if one chooses the left child of the distributional node n_{11}. Consequently, $q_{\text{RBON}}(\widehat{\mathcal{P}}_{\text{PER}}) = 0.75$.*

The complexity of join-free queries. Before reviewing the complexity of query answering for the aforementioned query languages, we make some preliminary remarks on complexity classes. Note first that for XML all problems are *decision* problems, i.e., to decide whether a query matches a document. In contrast, for p-documents all problems are *computational*, i.e., to compute the probability value. We thus need computational complexity classes such as FP (resp., FPSpace), the class of computational problems that can be solved by a Turing machine with output tape in polynomial time (resp., polynomial space), instead of the decision classes PTime or PSpace. The class #P is the class of computational problems that can be computed by counting the number of accepting runs of a nondeterministic polynomial-time Turing machine. Following [8], we say that a function is $\mathsf{FP}^{\#\mathsf{P}}$-hard if there is a polynomial-time *Turing reduction* (that is, a reduction with access to an oracle to the problem reduced to) from every function in $\mathsf{FP}^{\#\mathsf{P}}$ to it. Hardness for #P is defined in a standard way using Karp (many-one) reductions. For example, the function that counts for a propositional 2-DNF formula its number of satisfying assignments is #P-complete.We note that the use of Turing reductions in the definition of $\mathsf{FP}^{\#\mathsf{P}}$-hardness implies that any #P-hard problem is also $\mathsf{FP}^{\#\mathsf{P}}$-hard. Therefore, to prove $\mathsf{FP}^{\#\mathsf{P}}$-completeness it is enough to show $\mathsf{FP}^{\#\mathsf{P}}$-membership and #P-hardness.

We summarize in Table 1 the results that were obtained in the literature on the complexity of query answering of TMSO and TP queries over documents and p-documents. In terms of data complexity, evaluating a Boolean query over a document or a p-document can be done in linear time; this is a consequence of the formulation of TMSO in terms of tree automata [25], together with the possibility of coding a p-document as a probabilistic tree automaton [8, 4]. In terms of combined complexity, all computation can be made in polynomial space [4] and TMSO evaluation is PSpace-hard [20]. For tree-pattern queries, the situation is more interesting: they can also be evaluated linearly in the query size [5] on XML documents, but they become intractable over p-documents [17] under combined complexity.

3. QUERIES WITH VALUE JOINS

We explain now how to add joins to the query languages we have presented in the previous section. We motivate our study of these query languages with joins by noting that adding value joins to the language dramatically increases the complexity of query answering over probabilistic document.

We want to extend the ability of query languages on trees by allowing value joins, i.e., allowing to test for equality of the labels of nodes of the document. This is a very useful feature of query languages on trees, available in full in XPath 2.0, and, in a restricted form in XPath 1.0.

Joins add non-locality to the query language: it becomes necessary to remember the values of some nodes to compare them with the values of nodes elsewhere in the document. As we shall see, this has for consequence that tree-automata based techniques and their corresponding linear algorithms for query evaluation are no longer possible. Thus, joins cannot be expressed with a regular TMSO query; one could try to write some disjunction of Label_l predicates, but in the general case of an infinite set of values this would require infinitely many of them. For instance, to test that nodes n and m have the same value: $\bigvee_{v \in \mathcal{L}} (\mathsf{Label}_v(n) \wedge \mathsf{Label}_v(m))$.

Therefore, in order to express joins in MSO, an extra binary predicate $\mathsf{SameL}(\cdot, \cdot)$ is required, whose interpretation consists of pairs of nodes which have the same label, as defined by the preceding infinite disjunction. Since the joins considered are *value joins*, we further require that nodes whose labels are compared using the SameL predicate are document leaves, not internal nodes. Most results presented here extend to comparison of labels of internal nodes as well, with the exception of the dichotomy that we obtain in Section 5, as discussed there. The extension of TMSO with SameL is denoted TMSOJ. Similarly, TFOJ is the extension of TFO with SameL.

The language of tree-pattern queries with joins, TPJ, can be defined similarly but it is simpler to allow in the TP language a variable to be used multiple times. However, since we also consider only value joins, a variable used multiple times necessarily refers to a leaf in the documents and may consequently only appear as a leaf of the query. The class $\text{TPJ}^{\{/,[]\}}$ consist of TPJ queries without descendant edges.

EXAMPLE 6. *Consider the query q_{SBON} in Figure 1. The query asks whether Rick and Mary have received a bonus of the same value x. Clearly, q_{SBON} is true in d_{PER} since Rick and Mary both received a bonus of 44. Evaluation of q_{SBON} over $\widehat{\mathcal{P}}_{\text{PER}}$ returns true in only one world of $[\![\widehat{\mathcal{P}}_{\text{PER}}]\!]$, d_{PER}, since in all other worlds*

inria-00591905, version 1 - 10 May 2011

	TMSO		TP	
	Data	Combined	Data	Combined
XML	$O(\|d\|)$ [25]	PSpace-complete [20]	$O(\|d\|)$ [25]	$O(\|d\| \times \|q\|)$ [5]
PrXML^{mux,det}	$O(\|\widehat{\mathcal{P}}\|)$ [8]	FPSpace-complete [4]	$O(\|\widehat{\mathcal{P}}\|)$ [17]	FP$^{\#\mathsf{P}}$-complete [17]

Table 1: Complexity of query evaluation for join-free queries, with d a document, q a query, and $\widehat{\mathcal{P}}$ a p-document

either the first person is not Rick, or there is no bonus of the same value for Rick and Mary: $q_{\text{SBON}}(\widehat{\mathcal{P}}_{\text{PER}}) = \mathsf{Pr}(d_{\text{PER}}) = 0.4725$.

Our interest for joins comes from the following observation:

FACT 7 (LEMMA 9 OF [1]). *There is a Boolean* TPJ *query with* #P*-hard data complexity over* PrXML^{mux,det}.

Recall that for TP, and, indeed, for all MSO queries, the same problem is linear in the data size. Thus, adding joins to the language significantly increases the complexity of query evaluation. In the next sections we have a closer look at this problem.

4. TREE-MSO QUERIES WITH JOINS

In the previous section we saw that joins in tree-pattern queries come with a high cost: worst-case data complexity of querying p-documents goes from polynomial-time to #P-hard. In this section we investigate the cost of joins in TMSO queries.

Results of this section and the following one are summarized in Tables 2 (query evaluation) and 3 (deciding essential joins).

Querying p-documents. We first show that combined complexity of TMSOJ over p-documents remains as in the join-free case.

PROPOSITION 8. *Query evaluation for* TMSOJ *over* PrXML^{mux,det} *is* #P*-hard in data complexity and* FPSpace*-complete in combined complexity.*

PROOF. Hardness of data complexity comes form Fact 7. Hardness of combined complexity comes from the FPSpace-hardness of TMSO over PrXML^{mux,det} [4]. Let us now show the corresponding upper bound. Let $\widehat{\mathcal{P}}$ be a p-document and Q a TMSOJ query with n different variables. Since every TMSOJ query can be transformed in prenex normal form in polynomial time, we assume that Q is in such a form and Q' is its matrix, i.e., the quantifier-free part of Q. We describe an FPSpace algorithm to evaluate Q over $\widehat{\mathcal{P}}$. One enumerates all triples (d, ν, μ) with (1) a document $d \in [\![\widehat{\mathcal{P}}]\!]$; (2) an assignment ν of first-order variables of Q to nodes of d; (3) an assignment μ of second-order variables of Q to sets of nodes of d. For each triple one performs a polynomial time check whether $d, \nu, \mu \models Q'$.

This check is clearly polynomial time in the size of both the query and the document, since it boils down to checking that nodes of $\widehat{\mathcal{P}}$ given by ν and sets of nodes given by μ satisfy Ch, Label, and SameL conditions of Q'. The size of each triple (d, ν, μ) is polynomial in the size of both Q and d. Indeed, $|d|$ is bounded by $|\widehat{\mathcal{P}}|$, and assignments ν and μ are vectors (of length bounded by $|Q|$) of node identifiers or sets of node identifiers of d. □

Querying documents. The hardness of query evaluation for TMSOJ over probabilistic data is not surprising, since it is inherited from the hardness of TPJ queries. What is slightly more surprising is, as we immediately show, that query answering becomes intractable even over *deterministic data*.

PROPOSITION 9. *Query evaluation for* TMSOJ *over XML is* PSpace*-complete in combined complexity. Moreover, for each* $k \in \mathbb{N}$*, there are queries* q_1 *and* q_2 *of* TMSOJ *such that query evaluation of* q_1 *(resp.,* q_2*) over an XML document is* Π_k^P*-complete (resp.,* Σ_k^P*-complete).*

The combined complexity result is a direct consequence of the one for the join-free variant of the problem. To show the more interesting data complexity result, we present a way to encode arbitrary monadic second-order formulas on relational structures (MSO) into TMSOJ formulas on trees.

LEMMA 10. *There are two linear-time functions* f *and* g *s.t.*
(i) for every finite relational structure $\mathcal{A}$*,* $f(\mathcal{A})$ *is a document and*
(ii) for every sentence φ *of* MSO*,* $g(\varphi)$ *is a* TMSOJ *sentence such that* $f(\mathcal{A}) \models g(\varphi)$ *if and only if* $\mathcal{A} \models \varphi$.

PROOF. We start with f. See an example of $f(\mathcal{A})$ in Figure 2, left, where we encode the relation $R^{\mathcal{A}} = \{\vec{v}_1, \vec{v}_2\}$, for R of arity 2, and $\vec{v}_1 = (a, b)$, $\vec{v}_2 = (e, a)$. To generalize, let $\mathcal{A}$ be a finite structure over relations $R_1, \ldots, R_n$. Then $f(\mathcal{A})$ has a root labeled db with n children, one for each relation R_i, labeled R_i. If $R_i^{\mathcal{A}} = \{\vec{v}_1, \ldots, \vec{v}_m\}$, then the node labeled R_i has m children, one for each tuple $\vec{v}_j$, labeled $\vec{v}_j$. Every node labeled $\vec{v}_j$ has *arity*(R_i) children, where the k-th child is labeled k, for $k \in \{1, \ldots, \textit{arity}(R_i)\}$, and has only one child labeled with the value of the k-th attribute of $\vec{v}_j$. The described function f is obviously linear-time in the size of the input relation $\mathcal{A}$.

We now exhibit g. Let φ be a second-order logic sentence. We first preprocess φ. Let x be a join (first-order) variable of φ that has m occurrences. Then φ_x is φ where the i-th occurrence of x is substituted with a fresh variable w_i, quantification Qx, where $Q \in \{\exists, \forall\}$, is substituted with $Qw_1, \ldots, w_m$, and the resulting formula is conjuncted with the condition $\bigwedge_{i=1}^{m-1}(w_i = w_{i+1})$. Application of this transformation to all join variables of φ yields a formula φ' where all joins are "moved" to equality conditions and $\varphi \equiv \varphi'$.

Let $R(t_1, \ldots, t_n)$ be an atom of φ', such that R is not a second order variable in φ', where, w.l.o.g, the terms $t_1, \ldots, t_i$ are constants, and $t_{i+1}, \ldots, t_n$ are variables. Note that due to the preprocessing construction of φ' all the variables in $t_{i+1}, \ldots, t_n$ are different. Let $x, y, z, z_1, \ldots, z_n, w_{i+1}, \ldots w_n$ be variables that do not occur in φ'. One substitutes in φ' every occurrence of the atom $R(t_1, \ldots, t_n)$ with the following formula:

$$\psi_{R(t_1,\ldots,t_n)}(x, y_R) = \exists z_1 \ldots z_n \exists w_{i+1} \ldots w_n \ \mathsf{Label}_{\mathsf{db}}(x)$$
$$\wedge \mathsf{Ch}(x, y_R) \wedge \mathsf{Label}_R(y_R) \wedge \mathsf{Ch}(y_R, z) \wedge \left(\bigwedge_{j=1}^{n} \mathsf{Ch}(z, z_j) \wedge \mathsf{Label}_j(z_j) \right)$$
$$\wedge \left(\bigwedge_{j=1}^{i} \mathsf{Ch}(z_j, w_j) \wedge \mathsf{Label}_{t_j}(w_j) \right) \wedge \left(\bigwedge_{j=i+1}^{n} \mathsf{Ch}(z_j, t_j) \right),$$

where x is the same across all atoms of φ' and y_R is the same for all atoms with the predicate name R. If $R_1, \ldots, R_l$ are all the predicate names of φ', then one adds to the resulting formula the prefix $\exists x \exists y_{R_1} \cdots \exists y_{R_l}$. The next step is to substitute in the resulting formula every occurrence of the equality condition $(u = v)$ with the atom $\mathsf{SameL}(u, v)$, which yields φ''. Let ψ be an encoding

inria-00591905, version 1 - 10 May 2011

	TMSOJ		TPJ	
	Data	Combined	Data	Combined
XML	Σ_k^{P}-complete, Π_k^{P}-complete $\forall k \in \mathbb{N}$	PSpace-complete	PTime	NP-complete
PrXML^{mux,det}	#P-hard, in FPSpace	FPSpace-complete	FP$^{\#\mathsf{P}}$-complete	#P-hard, in FPSpace

Table 2: Complexity of query evaluation for queries with joins

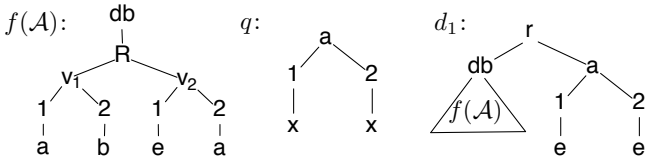


Figure 2: Left: translation from a finite relational structure $\mathcal{A} = \{R(a,b), R(e,a)\}$ into a tree $f(\mathcal{A})$ (Lemma 10). Center: a TPJ **query with an essential join (Theorem 11). Right: a document d (Theorem 11).**

of the tree structure that the function f above imposes on all $f(\mathcal{A})$. One can easily construct such a ψ. Finally, $g(\varphi) = \varphi'' \wedge \psi$.

The described transformation is obviously linear time and translates second-order logic formulas over arbitrary relations into TMSOJ formulas. By construction, $f(\mathcal{A}) \models g(\varphi)$ if and only if $\mathcal{A} \models \varphi$. □

Note that the function g from Lemma 10 applied to FO formulas returns TFOJ formulas. This property will be used later on to prove Theorem 11. We are now ready to prove Proposition 9.

PROOF OF PROPOSITION 9. Combined complexity has been discussed. As shown by Ajtai, Fagin, and Stockmeyer in [3] (Theorem 11.2) there are MSO queries over graphs whose evaluation is monadic Σ_k^{P}-complete for every k (monadic Σ_k^{P} is the class of MSO-expressible problems with a prefix of k alternations of *second-order predicates* starting with $\exists$, and an arbitrary first-order matrix, disregarding the number of alternations of first-order predicates); their negation is thus monadic Π_k^{P}-complete. This gives a Σ_k^{P} lower bound for MSO query evaluation over arbitrary structures. At the same time Lemma 10 allows to reduce the latter problem to the one of TMSOJ, which immediately gives us the lower bound for data complexity. The upper bound for data complexity follows from [24] where Stockmeyer showed that monadic Σ_k^{P} is in Σ_k^{P}. □

Deciding essential joins. As we saw, joins are expensive in TMSOJ for querying documents and p-documents. In contrast TMSO queries are tractable over both deterministic and probabilistic documents. What we study now is the problem of determining whether a TMSOJ query is essentially join-free and, consequently, can be evaluated efficiently.

THEOREM 11. *Deciding if a query has an essential join is undecidable for* TFOJ *and* TMSOJ.

We will prove this theorem by reduction from finite satisfiability for first order logic formulas, which is known to be undecidable [26].

The next lemma shows that TFO queries are insensitive to the multiplicity of constants that occur in the documents but not in the queries. We need the following notions. Let d be a document where a label a occurs k times. We denote by d^a a document obtained from d by replacing each occurrence of a with a distinct fresh constant. A *canonical document* d_q for a TPJ query q is a document that is obtained from q by replacing every descendant edge with a child edge and by replacing each different variable with a fresh constant.

Query Language	Deciding Joins
TMSOJ	undecidable
TFOJ	undecidable
TPJ	Π_2^{P}-complete
TPJ$^{\{/,[]\}}$	NP-complete

Table 3: Complexity of deciding essential joins

LEMMA 12. *Let q be a* TFO *or* TP *query, d a document and $a \in \mathcal{L}$ a label occurring in d multiple times, but not occurring in q. Then $d \models q$ if and only if $d^a \models q$.*

For TP queries the proof follows from the fact that only nodes of q labeled with variables can be mapped to the nodes of d labeled with a. For TFO queries the proof can be done by induction on q.

Now we can prove Theorem 11.

PROOF OF THEOREM 11. By reduction from finite satisfiability of first-order formulas over relational structures.

Let φ be an FO formula and $g(\varphi)$ be the corresponding TFOJ formula constructed as described in Lemma 10. Consider the TPJ query q in Figure 2, center; with a slight abuse of notation, we denote its TFOJ encoding also as q. Assume that q and $g(\varphi)$ have no common labels. It is easy to see that q has an essential join.

We now show that

1. If the TFOJ formula $(g(\varphi) \rightarrow q)$ has no essential joins, then φ is *not* finitely satisfiable.
2. If the TFOJ formula $(g(\varphi) \rightarrow q)$ has an essential join, then φ is finitely satisfiable.

Assume $g(\varphi) \rightarrow q$ has no essential joins, that is, $(g(\varphi) \rightarrow q) \equiv \psi$, where $\psi \in$ FO. If φ is finitely satisfiable, then there is a finite structure $\mathcal{A}$, s.t., $\mathcal{A} \models \varphi$. Let $d = f(\mathcal{A})$ be the document computed from $\mathcal{A}$ as described in Lemma 10. W.l.o.g. we can assume that d has no labels occurring in q. By this lemma, $d \models g(\varphi)$.

Consider a new document d_1, that is a combination of d and d_q, a canonical document of q, as in Figure 2, right, where a does not occur in d. Clearly $d_1 \models \psi$ holds and consequently by Lemma 12, $d_1^e \models \psi$ also holds. Since by construction $d_1^e \models g(\varphi)$, we conclude $d_1^e \models q$. At the same time, since q has a join variable, $d_1^e \not\models q$. We obtain a contradiction.

Assume $g(\varphi) \rightarrow q$ has an essential join. If φ is not finitely satisfiable, then, due to Lemma 10, we conclude that $g(\varphi)$ is not finitely satisfiable. Therefore, the implication $g(\varphi) \rightarrow q$ is a tautology, expressible without joins, and consequently the implication has no essential joins. We obtain a contradiction. □

The proof shows that the set of formulas with essential joins is not co-recursively enumerable. On the other hand, the set of finitely satisfiable FO formulas is recursively enumerable, but we do not know if this is also true of deciding essential joins: a TFOJ query q has essential joins iff *for every* TFO query q', *there exists* a finite tree modeling of one but not the other. This alternation of quantifiers does not lend itself to a straightforward enumerability proof.

inria-00591905, version 1 - 10 May 2011

5. TREE-PATTERN QUERIES WITH JOINS

In the previous section we showed that joins increase worst-case data complexity of TMSO, while they do not affect combined complexity. What we do not know is whether *all* queries in TMSOJ not in TMSO are hard, or whether there are some queries with essential joins that are still tractable. A TPJ query is a *basic join query* if it has exactly one join variable and this variable occurs exactly twice. As we now show, in the case of basic join queries, *every* query that is not equivalent to a query in TP is $\mathsf{FP}^{\#\mathsf{P}}$-hard for probabilistic documents.

Querying documents. We first study data and combined complexity of TPJ query evaluation over deterministic documents. Recall that TP query answering over XML is polynomial in combined complexity. The situation changes for TPJ: we now show that evaluation of TPJ queries over XML is essentially the same as querying relational structures with conjunctive queries.

PROPOSITION 13. *Query evaluation for* TPJ *over XML is* PTime *in data complexity and* NP*-complete in combined complexity.*

Our proof is based on the observation that TPJ over XML behaves in the same way as the class of conjunctive queries over arbitrary relational structures.

LEMMA 14. *Let* $\Sigma = \{\mathsf{Ch}, \mathsf{Desc}, \mathsf{Label}\}$. *Then there exist functions* f *and* g *computable in* PTime *such that*
(i) for every document d, $f(d)$ *is a relational structure over* Σ *and*
(ii) for every TPJ *query* q, $g(q)$ *is a conjunctive query over* Σ,
such that $d \models q$ *if and only if* $f(d) \models g(q)$.

PROOF OF PROPOSITION 13. Lemma 10 restricted to conjunctive queries gives their encoding in TPJ and the NP-hardness of combined complexity, since the combined complexity of query evaluation over relational structures is NP-complete for conjunctive queries [6]. NP-membership holds since one can guess a mapping from the nodes of a TPJ query to the nodes of a document and check in polynomial time that the mapping is a valuation.

It is known that data complexity of query evaluation for conjunctive queries is in PTime [15]. Combining this result with Lemma 14 gives the data complexity for TPJ. □

Gottlob et al. in [13] studied conjunctive queries over trees that are related to TPJ. In their setting joins can be done on identical nodes only, while we can join arbitrary nodes as long as they carry the same label. As we showed in Lemma 9, this allows us to encode queries about arbitrary relational structures into our query model and data model. Therefore, our Proposition 13 follows from well-known results about relational conjunctive queries, which is not true for the results in [13].

Querying p-documents. We show that for basic join queries, there is a dichotomy between tractable and intractable queries. It is promising that this dichotomy has a very simple characterization, even more so when it is contrasted with the dichotomy of conjunctive queries over tuple-independent probabilistic databases [10], where the condition for hardness is much more involved.

THEOREM 15 (DICHOTOMY). *For every basic join query* q, *evaluation over* $\mathsf{PrXML}^{mux,det}$ *is*

- *feasible in time linear if* q *is essentially join-free;*
- $\mathsf{FP}^{\#\mathsf{P}}$*-complete in data complexity otherwise.*

To show the data complexity upper bound we need the following lemma, that can be proved by adopting the techniques developed by Grädel, Gurevich, and Hirsch in [14].

LEMMA 16. *Let* $\mathcal{Q}$ *be a query language with polynomial-time data complexity over XML. Then* $\mathcal{Q}$ *is of* $\mathsf{FP}^{\#\mathsf{P}}$ *data complexity over* $\mathsf{PrXML}^{mux,det}$.

PROOF OF THEOREM 15. Lemma 16 and Proposition 13 give the upper bound, while an extension of the proof for Lemma 9 [1] gives hardness by reduction from #2-DNF satisfiability.

We first exhibit a #P-hard TPJ query q and then show how to generalize the construction to TPJ queries with just one variable that occurs twice. Consider the TPJ query q in Figure 3 (left), that clearly has an essential join.

Consider an encoding of the 2-DNF formula:

$$\varphi = (w_1 \wedge v) \vee (\neg v \wedge w_2) \vee (\neg v \wedge w_3) \vee (w_4 \wedge \neg v)$$

as a p-document $\widehat{\mathcal{P}}_\varphi$ in Figure 3 (left) that one can immediately generalize to arbitrary 2-DNF formulas. Observe that every $d \in [\![\widehat{\mathcal{P}}_\varphi]\!]$ has the same probability, say p, and $\mathsf{Pr}(\widehat{\mathcal{P}}_\varphi \models q) = p \times n$, where n is the number of satisfying assignments for φ. Thus, answering TPJ queries over $\mathsf{PrXML}^{mux,det}$ is #P-hard.

We now show the #P-hardness using the same kind of reduction as for the query q above. Let q be a TPJ query with one variable x occurring twice. To distinguish these two occurrences we refer to them as x_1 (labeling a node n_1 of q) and x_2 (labeling n_2). We now exhibit a p-document $\widehat{\mathcal{P}}'_\varphi$ such that computation of $\mathsf{Pr}(\mathcal{P}'_\varphi \models q)$ is #P-hard. $\widehat{\mathcal{P}}'_\varphi$ is composed of two parts: the first one, $\widehat{\mathcal{P}}''_\varphi$, is for the subqueries q_{x_1} and q_{x_2} of q' related to x_1 and x_2, and the second one, d, for the remaining part of q', that is, q' without q_{x_1} and q_{x_2}. We now present q_{x_i}'s, then $\widehat{\mathcal{P}}''_\varphi$ and finally d.

Let r be the root of q. Since in TPJ queries join variables label only leaves, there are two paths in q: from r to the leaf x_1 and from r to x_2. These paths may share some nodes, thus, assume the node m labeled a is the least common ancestor of x_1 and x_2, see Figure 3 (right). Let q_{x_1} be the maximal subquery of q such that *(i)* its root is the child of a, *(ii)* its root is between a and x_1. A query q_{x_2} for x_2 is defined analogously.

Now observe that a cannot be a parent of both x_1 and x_2 in q, otherwise x is not an essential join. Indeed, if a is the parent of both x_1 and x_2, then by deleting one of the nodes labeled x_i one obtains a query equivalent to q, which contradicts the essentialness of x. Let d_1 and d_2 be documents such that $d_1 \models q_{x_1}$ while $d_1 \not\models q_{x_2}$, and $d_2 \models q_2$ while $d_2 \not\models q_1$. Such d_i's always exist and can be constructed in EXPTime in the size of the q_i's due to Proposition 3 of [21].

We modify d_1 further as d'_1 by adding a child node with a fresh label to all leaf nodes of d_1 except for one given homomorphic image u_1 of x_1 from q_{x_1} into d_1. This will ensure that the value join will necessarily involve this specific node and not another one of d_1, thanks to our condition that joins only match document leaves. We transform similarly d_2 into d'_2.

Assume a is a proper ancestor (ancestor but not a parent) of either x_1 or x_2. Consider the p-document $\widehat{\mathcal{P}}''_\varphi$ obtained from $\widehat{\mathcal{P}}_\varphi$ by (i) relabeling the root with a, (ii) substituting every node labeled l with the document d'_1 (if q_1 is empty, then l is substituted with a *det* node), and (iii) every node labeled r with d'_2 (if q_2 is empty, then r is substituted with a *det* node). Here, substituting a node labeled l with d'_1 means the following. Let node n be labeled l. Then (i) a copy of d'_1 is inserted below the parent of n (that is, n is replaced with d'_1); (ii) for the homomorphic image u_1 of x_1 that has been previously chosen, we insert a new *det* node into d'_1 as a sibling of m_i; and (iii) copy all children of n below that *det* node. In a similar fashion, we substitute r with d'_2.

Let the query q' be a obtained from q by deleting q_{x_1} and q_{x_2} together with x_1 and x_2. Since x is essential in q, we have $q' \not\sqsubseteq q$,

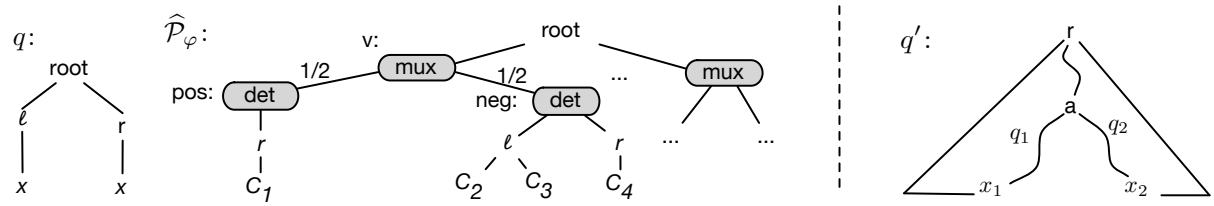


Figure 3: For Theorem 15. Left: a TPJ **query** q **with an essential join and a p-document** $\widehat{\mathcal{P}}_\varphi$ **encoding** $\varphi = (w_1 \wedge v) \vee (\neg v \wedge w_2) \vee (\neg v \wedge w_3) \vee (w_4 \wedge \neg v)$**. Right: a general pattern** q' **of a** TPJ **query with an essential join.**

hence, there is a document d such that $d \models q'$ and $d \not\models q$. Let ν be a homomorphism from q' to d and $k = \nu(m)$.

Finally, the p-document $\widehat{\mathcal{P}}'_\varphi$ is obtained from d by inserting at the node k the subtrees of $\widehat{\mathcal{P}}''_\varphi$ rooted at its root. Observe that by construction of $\widehat{\mathcal{P}}'_\varphi$, the probability of every $d' \in [\![\widehat{\mathcal{P}}'_\varphi]\!]$ is again p.

We now show that $\mathsf{Pr}(\widehat{\mathcal{P}}'_\varphi \models q') = p \times n$, where n is the number of satisfying assignments for φ. Indeed, observe that in the query composed of q_{x_1} and q_{x_2} rooted at m x is an essential join. Moreover, this query can not be homomorphically embedded in d (due to construction of d) and in a document $d'' \in [\![\widehat{\mathcal{P}}''_\varphi]\!]$ in a way that none of x_1 and x_2 is mapped to C_i, for some clause i (due to the fact that x is essential). This gives us that for every $d' \in [\![\widehat{\mathcal{P}}'_\varphi]\!]$: μ is a homomorphism from q to d' iff the label of $\mu(n_1) = \mu(n_2) = C_i$, for some clause i. Thus, by construction of $\widehat{\mathcal{P}}'_\varphi$, there is a bijection χ between the set of worlds of $[\![\widehat{\mathcal{P}}'_\varphi]\!]$ that satisfy q and the set M of satisfying assignments of φ, which yields:

$$\mathsf{Pr}(\widehat{\mathcal{P}}'_\varphi \models q) = \sum_{\mu \in M} \mathsf{Pr}(\chi(\mu)) = \sum_{\mu \in M} p = p \cdot |M|,$$

and concludes the proof. □

It is open whether this dichotomy theorem can be extended to the general case of multiple variables with possibly more than two occurrences. Another important limitation of this result is that it only holds when join variables are required to be on query leaves (*value joins*). It is for instance easy to see that if two join variables label nodes in a parent-child relationship, the query can be evaluated efficiently even if the join is essential. This comes from the fact that given a p-document node, it is possible to deterministically get the label of its parent. In a more general setting where the label of internal nodes can be given by a probability distribution, the dichotomy proof can be adapted.

Now we know that essential joins are an important criterion to determine whether queries are intractable. Thus, it is important to be able to detect whether a TPJ query is essentially join-free. In the next section we present a conceptually simple test for essential joins. Unfortunately, as we also show, this test is intractable.

Deciding essential joins. We define the *core* of a TPJ query q, denoted $\mathsf{cr}(q)$, as the TP query obtained from q by replacing every occurrence of a join variable with a distinct fresh variable.

THEOREM 17. *A* TPJ *query* q *has no essential joins iff* $q \equiv \mathsf{cr}(q)$.

To prove this we use the following fact due to Miklau and Suciu [21]. Let q be a TPJ query and n the length of q's longest chain of nodes, where all nodes are labeled with wildcards (i.e., non-join variables) and all edges are child edges. Let $a \in \mathcal{L}$ be a label that does not occur in q and $D_{a,q}$ is the set of documents obtained from q by (i) expanding every descendant-edge into a chain of child-edges of length at most $n+1$, (ii) labeling every node of these chains with a, and (iii) substituting every variable with a fresh label.

FACT 18 ([21]). *Let* q_1 *and* q_2 *be* TP *queries and let* a *be a label that does not occur in* q_1 *and* q_2*. Then* $q_1 \sqsubseteq q_2$ *if and only if* $d \models q_2$ *for every document* $d \in D_{a,q_1}$.

PROOF OF THEOREM 17. The *if* direction is obvious. For *only-if*, assume $q \equiv \hat{q}$ for a TP query $\hat{q}$. We show that $q \equiv \mathsf{cr}(q)$. Since $q \sqsubseteq \mathsf{cr}(q)$ obviously holds, we will prove $\mathsf{cr}(q) \sqsubseteq q$.

Assume that q has exactly one join variable, that is, there is exactly one variable x occurring in q more than once. Let N be all nodes in q labeled with x.

Let a and b be fresh constants for both q and $\hat{q}$. Consider the set of documents $D_{a,q}$, where, w.l.o.g., we assume that in every $d \in D_{a,q}$ all the nodes of N are labeled with b. By construction, for every $d \in D_{a,q}$ we have $d \models q$. Hence, $d \models \hat{q}$ also holds and, by Lemma 12, we obtain $d^b \models \hat{q}$. Let us collect all such d^b's in $D^b_{a,q}$, that is, $D^b_{a,q} = \{d^b \mid d \in D_{a,q}\}$. By construction $D^b_{a,q} = D_{a,\mathsf{cr}(q)}$, and we are in the conditions of Fact 18: $q_1 = \mathsf{cr}(q)$ and $q_2 = \hat{q}$ are two TP queries such that for every $d \in D_{a,\mathsf{cr}(q)}$ it holds $d \models \hat{q}$. Hence, $\mathsf{cr}(q) \sqsubseteq \hat{q}$ and, due to equivalence of q and $\hat{q}$, we obtain $\mathsf{cr}(q) \sqsubseteq q$.

The proof can be extended to the general case, when q has more than one join variable, by iterating the construction above over all join variables. □

We have a conceptually simple test for essential joins: it is sufficient to test that a query is equivalent to its core to guarantee that it is essentially join-free. The next theorem shows that this test is expensive for the class of queries with at least child navigation and branching.

THEOREM 19. *Deciding if a query has an essential join is* Π_2^P*-complete for* TPJ *and* NP*-complete for* $\mathsf{TPJ}^{\{/,[]\}}$.

A proof can be done using similar construction as by Deutsch and Tannen in [11] for the problem of query containment for XPath extensions.

6. CONCLUSION AND DIRECTIONS

We studied complexity of query evaluation over XML and probabilistic XML for tree-pattern and monadic second-order queries with joins. We also investigated the complexity of deciding essential joins. Our results are summarized in Tables 2 and 3.

There are a number of open questions remaining in our study: a tight complexity bound for combined complexity of TPJ over PrXML, semi-decidability of essential joins for TMSOJ, and a criterion for deciding essential joins in TMSOJ.

Another major open question is whether the dichotomy of tractability for basic TPJ queries extend to arbitrary ones, still relying on the notion of essential join. If it does, we have a remarkable contrast with what happens for relational probabilistic data. Our dichotomy is conceptually very simple: it is sufficient to test that a query is equivalent to its core to guarantee that it is tractable, but this test itself is intractable in the query size. In the relational

inria-00591905, version 1 - 10 May 2011

inria-00591905, version 1 - 10 May 2011

setting, Dalvi and Suciu proved a dichotomy for conjunctive queries [10] over *block-independent databases* which guarantees tractability of queries, but involves a conceptually complicated, though polynomial-time testable, characterization of queries. We would like to understand better the connections between these dichotomy results, if any.

Continuing with the discussion after the proof of Theorem 15 on the dichotomy, another further direction to study is a more general form of joins in queries. In our query model we allow for joins on document leaves only, while one may also think of *structural joins*, that is, joins which can be imposed on the labels of *intermediate nodes* in trees and not only on the leaves. One way of preserving the results of this paper in the setting of both value and structural joins is to consider a slightly different data model. The current model allows for distributional nodes that define *structural* probabilistic alternatives, that is, it allows for distributions on children of nodes only. What we need are *non-structural* probabilistic alternatives, that is, distributional nodes that define alternatives of labels for a given node. For example, we should be able to express that a node n of a document is labeled either with a or b.

Another direction is a more general study of joins in the context of both queries and data. Recall, when queries and probabilistic data are tree-shaped, query answering is tractable, while adding joins to queries makes it hard. An analogous situation happens when we add a form of joins in data, as it is done in [2], where probabilistic XML is defined by means of probabilistic annotations (that are conjunctions of literals over Boolean random variables) on the nodes of documents. For such probabilistic data query answering becomes hard for tree-shaped queries: every TP query is either trivial, i.e., it retrieves just the root of the data, or it is $\mathsf{FP}^{\#\mathsf{P}}$-complete [17], and we again have a dichotomy. Note that the probabilistic data of [2] is tree-shaped, but the dependencies between the annotations are graph structured. Another way of introducing joins in data is by considering data graphs, where MSO query answering is Σ_k^{P}-complete [3]. To sum up, joins can be seen as a way to add graph structure either to queries, or to data (on the level of data itself or on the level of probabilistic dependancies), and we are missing a study that bridges these alternatives.

7. REFERENCES

[1] S. Abiteboul, T.-H. H. Chan, E. Kharlamov, W. Nutt, and P. Senellart. Aggregate queries for discrete and continuous probabilistic XML. In *ICDT*, 2010.

[2] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB J.*, 18(5):1041–1064, 2009.

[3] M. Ajtai, R. Fagin, and L. J. Stockmeyer. The closure of monadic NP. *J. Comput. Syst. Sci.*, 60(3), 2000.

[4] M. Benedikt, E. Kharlamov, D. Olteanu, and P. Senellart. Probabilistic XML via Markov chains. *PVLDB*, 3(1):770–781, 2010.

[5] M. Benedikt and C. Koch. XPath leashed. *ACM Comput. Surv.*, 41(1), 2008.

[6] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC*, 1977.

[7] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of Web information extraction systems. *IEEE TKDE*, 18(10), 2006.

[8] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

[9] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *CACM*, 52(7), 2009.

[10] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.

[11] A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In *Proc. KRDB*, 2001.

[12] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. *VLDB J.*, 18(2), 2009.

[13] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.

[14] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *Proc. PODS*, 1998.

[15] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[16] E. Kharlamov, W. Nutt, and P. Senellart. Value joins are expensive over (probabilistic) XML (extended version). Technical Report KRDB11-1, KRDB Research Centre, Free University of Bozen-Bolzano, 2011.

[17] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic XML. *VLDB J.*, 18(5):1117–1140, 2009.

[18] C. Koch. MayBMS: A system for managing large uncertain and probabilistic databases. In C. Aggarwal, editor, *Managing and Mining Uncertain Data*. Springer, New York, NY, 2009.

[19] J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, San Fransisco, CA, 2001. Morgan Kaufmann.

[20] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[21] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1), 2004.

[22] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proc. VLDB*, 2002.

[23] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 2001.

[24] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[25] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*. Springer-Verlag, 1997.

[26] B. A. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *American Mathematical Society Translations Series 2*, 23, 1963.

[27] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, pages 459–470, 2005.

[28] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.

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