Containment of Partially Specified Tree-Pattern Queries

Dimitri Theodoratos
dth@cs.njit.edu
NJIT, Newark, USA

Theodore Dalamagas
dalamag@dblab.ece.ntua.gr
NTU Athens, Greece

Pawel Placek
pp58@njit.edu
NJIT, Newark, USA

Stefanos Souldatos*
stef@dblab.ece.ntua.gr
NTU Athens, Greece

Timos Sellis
timos@dblab.ece.ntua.gr
NTU Athens, Greece

Abstract

Nowadays, huge volumes of data, including scientific data, are organized or exported in tree-structured form. Querying capabilities are provided through tree-pattern queries. The need for integrating multiple data sources with different tree structures has driven, recently, the suggestion of query languages that relax the complete specification of a tree pattern.

In this paper we adopt a query language with partially specified tree-pattern queries. A central feature of this type of queries is that the structure can be specified fully, partially, or not at all in a query. Important issues in query optimization require solving the query containment problem. We study the containment problem for partially specified tree-pattern queries. To support the evaluation of such queries, we use semantically rich constructs, called dimension graphs, which abstract structural information of the tree-structured data. We address the problem of query containment in the absence (absolute query containment) and in the presence (relative query containment) of dimension graphs, and we provide necessary and sufficient conditions for each type of query containment. We suggest a technique for relative query containment checking based on structural information extracted in advance from the dimension graph. Our approach is implemented and validated through extensive experimental evaluation.

1. Introduction

Nowadays, huge volumes of data, including scientific data, are organized in a tree structured form. Even if data is not stored natively in tree structures, export mechanisms make data publicly available in that form to enable its automatic processing by programs, scripts, and agents on the Web. Data encoded with the XML is by far the most prominent example. Queries on tree-structured data are mainly based on tree patterns. These tree patterns allow child and descendant relationship edges. XPath [1] is a language that uses tree patterns (branching path expressions) to navigate through the tree structure of an XML document, and lies at the core of W3C language proposals for XML querying (e.g. XQuery [2]) and transformation (e.g. XSLT [3]).

In this context, perhaps the most challenging problems are the integration of data sources with different tree structures, and the querying of data sources when their structure is not fully known to the user. Previous approaches to the tree-structured data source integration issue range from traditional mapping-rule approaches between a global structure and local structures [7], to structure-less keyword-based approaches [10, 8], to approximate approaches [5, 9]. Their failure to provide efficient answers when applied to the problems above has motivated recently the proposal of query languages that combine keyword-based search and structured querying [12]. In [17, 18] we introduce a query language that provides partial specification of a tree-pattern. A central feature of this language is that the structure can be specified fully, partially, or not at all in a query while returning structured answers.

In contrast to traditional query languages for tree-structured data where the query pattern is a tree (possibly involving double-line edges, i.e. ancestor-descendent relationships), there is not necessarily a total order for the nodes in a "path" of a partially specified tree pattern. Precedence relationships (ancestor-descendent and parent-child relationships) are optional in them. We call those paths partially specified paths (PSPs). Different PSPs are connected through expressions indicating shared nodes. This flexibility in specifying queries allows addressing the aforementioned problems efficiently, because only those precedence

*The author partially supported by the Alexander S. Onassis Public Benefit Foundation scholarship.

relationships that are known to hold on the data sources or that are required need to be specified in a query. On the other hand, processing these queries becomes more involved because new precedence relationships and node sharing expressions can be derived from those explicitly specified in a query [18].

The problem. A query language needs to be complemented with query processing and optimization techniques. Important issues in query optimization, including query satisfiability [11], query minimization [4, 19, 16], and query rewriting using views [15], require solving the query containment problem. In this paper, we address query containment for partially specified tree structured queries. This problem has been studied in the past for (fully specified) tree pattern queries in the absence [4, 13] and in the presence [6, 20, 14] of constraints. To the best of our knowledge, it has not been addressed in the context of partially specified tree patterns. Most of the previous work focuses almost exclusively on characterizing the complexity of the containment problem. Our goal here is not to provide complexity results but efficient methods for checking query containment that can be used for processing partially specified tree-pattern queries.

Contribution. We consider trees of values whose nodes are partitioned to form what we call dimensions. Queries are specified on dimensions annotated with values and are not cast on the structure of a specific value tree. The dimensions are used to define dimension graphs, a construct that abstracts the structural information of the value trees. The dimension graphs can be automatically extracted from value trees and support the processing and evaluation of the queries. Using a dimension graph $\mathcal{G}$, we can identify, for each partially specified tree pattern query Q, a set of (fully specified) tree pattern queries that are "equivalent" to Q in that they can be used to compute the answer of Q on any value tree underlying $\mathcal{G}$. We call these fully specified queries "dimension trees", and we use them to evaluate partially specified tree pattern queries over value trees.

The main contributions of this paper are the following:

- We define two types of query containment: absolute query containment and query containment with respect to a dimension graph (relative query containment). Relative query containment holds on value trees that underlie a given dimension graph.
- In order to allow query comparison we define a "normal form" for queries, called full form. Intuitively, a full form of a query comprises all the precedence relationships and annotated dimensions that can be derived from those specified in the query.
- We provide necessary and sufficient conditions for absolute query containment in terms of query homomorphisms and for relative query containment in terms of absolute containment of dimension trees.
- For coping with the high complexity of the relative containment we introduce a technique for this type of containment which is sound but not complete.
- We have implemented our approach. An experimental evaluation compares the time required for the absolute and relative containment checking, and shows that our technique importantly reduces the time for checking relative query containment.

Outline. The next section overviews the data model and the query language. Section 3 provides definitions and related concepts. In Section 4, we present conditions for absolute and relative query containment. Experimental results are shown in Section 5. We conclude in Section 6 and discuss future work. Due to lack of space, proofs are omitted.

2. Data Model and Query Language

We present in this section our data model and query language initially introduced in [18]. The data model represents tree-structured data using the concepts of value tree, dimension and dimension graph. The query language allows for partially specifying tree patterns.

2.1. Value Trees and Dimension graphs

We assume an infinite set of values V that includes a special value r.

Dimensions and value trees. A *dimension set* over V is a partition $\mathcal{D}$ of V that includes the singleton $\{r\}$. Each element of $\mathcal{D}$ is called *dimension* of $\mathcal{D}$. The dimensions in $\mathcal{D}$ are assigned distinct names. In particular, the dimension $\{r\}$ is named R. Intuitively, a dimension is a set of semantically related values. For instance, $Measure$ can be a dimension that includes values `Temperature` and `Humidity`. Since the names of the dimensions are distinct we use them to identify the dimensions of $\mathcal{D}$. Different applications may require and apply different partitions of the values in V. For the needs of this paper, we assume that a dimension set is fixed and we denote it by $\mathcal{D}$.

Definition 2.1 A *value tree* over $\mathcal{D}$ is a rooted node-labeled tree T, such that: (a) Each node label in T belongs to V, (b) Value r labels only the root of T, and (c) There are no two nodes on a path in T labeled by values that belong to the same dimension in $\mathcal{D}$. □

Example 2.1 Let $\mathcal{D} = \{Root, Period, Measure, Location, Format, Season\}$. Figure 1 shows a value tree T. This value tree includes meteorological information concerning temperature and humidity for European cities. Such a tree encodes meteorological information from multiple data sources, possibly with structural inconsistencies. For

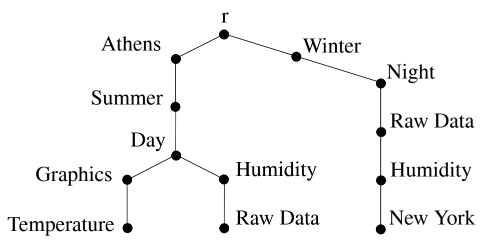

Figure 1. Value tree T

Dim.	Abbr.	Values
Root	R	{r}
Period	A	{Day, Night}
Measure	B	{Temperature, Humidity}
Location	C	{Athens, New York}
Format	D	{Raw Data, Graphics}
Season	E	{Summer, Winter}

Figure 2. Dimension set $\mathcal{D}$

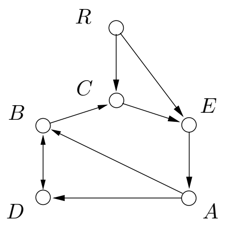

Figure 3. Dimension graph $\mathcal{G}$

example, measures are categorized by their type and then by their presentation format (e.g., `Humidity` and `Raw Data` for Athens) or the other way around (e.g., `Raw Data` and `Humidity` for New York).

Two dimensions can have their values appearing in different order in different paths of the value tree. This is the case of dimensions $Measure$ and $Format$, or of dimensions $Location$ and $Period$ in value tree T.

The dimensions of set $\mathcal{D}$, abbreviated as shown in Figure 2.1, are used for the rest of the examples in this paper. □

The semantic interpretation of the values of a value tree into dimensions is provided by a user, possibly assisted by an ontology. Note, however, that dimensions can also be chosen to represent purely syntactic objects, e.g. the elements of an XML document.

Dimension graphs. The values of some dimension may not be children or descendants of any value of some other dimension in a value tree. For instance, no value of dimension $Period$ in the value tree T of Figure 1 is a child of a value of a dimension other than $Season$. We use the concept of dimension graph to capture this type of relationship between dimensions in a value tree.

Definition 2.2 Let T be a value tree over $\mathcal{D}$. A *dimension graph* of T is a graph (N, E), where N is a set of nodes and E is a set of edges defined as follows: (a) There is a node D in N if and only if there is a value in T that belongs to dimension D, and (b) There is a directed edge (D_i, D_j) in E if and only if there are nodes n_i and n_j in T labeled by values $v_i \in D_i$ and $v_j \in D_j$, respectively, such that n_j is a child n_i in T. If $\mathcal{G}$ is a dimension graph of a value tree T, we say that T *underlies* $\mathcal{G}$. □

The dimension graph of a value tree may have cycles. In particular, it can have a trivial cycle if, in the underlying value tree, a value of a dimension labels a parent node of a node labeled by a value of another dimension and conversely.

Example 2.2 Figure 3 shows the dimension graph of the value tree T of Figure 1. Trivial cycles are shown in the figures with a double headed edge (e.g. the edge between dimensions $Format$ and $Measure$ in Figure 3). □

Dimension graphs can be automatically extracted from value trees and abstract their structural information. As we show in subsequent sections, they help the evaluation of queries on value trees, the detection of unsatisfiable queries and the checking of query containment.

2.2. Query Language

Queries are issued on dimension sets and are evaluated on values trees. Dimension graphs can support the formulation of queries. To allow query composition, we require that the evaluation of a query on a value tree yields a value tree.

Syntax. A query on a dimension set provides a (possibly partial) specification of a tree of dimensions annotated with sets of values. The tree is rooted at dimension R. A query specifies such a tree through a set of (possibly partially specified) paths from the root of the tree. For succinctness, in the following, *PSP* stands for *partially specified path*. Each PSP is defined in a query by a set of annotated dimensions, and a set of precedence relationships (child and descendent relationships) among these annotated dimensions. A dimension can be annotated by a set of values or a question mark denoting any value of the dimension. A query further indicates nodes (annotated dimensions) that are shared among different PSPs in the query tree. It also identifies a distinguished PSP called output PSP.

Before stating the formal definitions, we provide some intuition using a graphically represented query.

Example 2.3 Figure 4 shows a query. This query searches for daylight measures, in all available formats, concerning Athens. Note that we do not specify any order between dimensions $Measure$ and $Format$ in PSP p_2 (the output path). Thus, a value of $Measure$ might be an ancestor or a descendant of a value of $Format$. However, both values have to be in the same path and descendants of the value of $Period$. Similarly, no order is specified between $Period$

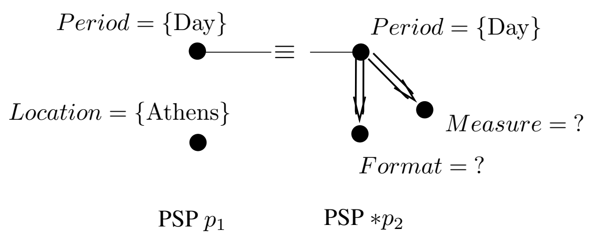
$Period = \{\text{Day}\}$
$\equiv$
$Period = \{\text{Day}\}$
$Location = \{\text{Athens}\}$
$Measure = ?$
$Format = ?$
PSP p_1
PSP $*p_2$

Figure 4. Query Q

and $Location$ in PSP p_1. However, their values have to be on the same path. The value for $Location$ (`Athens`) can be on the same or on different paths with the value of $Format$ and $Measure$. The value `Day` of $Period$ should be a common (shared) node of the two paths represented by the two PSPs. □

Definition 2.3 A *query* on a dimension set $\mathcal{D}$ is a triple $(\mathcal{P}, \mathcal{S}, o)$, where:

(a) $\mathcal{P}$ is a nonempty set of triples $(p, \mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ and $\mathcal{R}$ define a PSP as explained below, and p is a distinct name for this PSP. Since PSP names are distinct, we identify PSPs with their names.

(a1) $\mathcal{A}$ is a set of expressions of the form $D[p] = V$, where $D[p]$ denotes the dimension D of $\mathcal{D}$ in PSP p, and V is a set of values of dimension D or a question mark ("?"). These expressions are called *annotating expressions* of p. If the expression $D[p] = V$ belongs to $\mathcal{A}$ we say that D is *annotated* in p and V is its *annotation*. A dimension can be annotated only once in a PSP p. Without mentioning it explicitly, we assume that dimension R is annotated with a '?' in every PSP. Set $\mathcal{A}$ can be empty.

(a2) $\mathcal{R}$ is a set of expressions of the form $D_i[p] \rightarrow D_j[p]$ or $D_i[p] \Rightarrow D_j[p]$, where D_i is an annotated dimension in $\mathcal{A}$ or R, and D_j is an annotated dimension in $\mathcal{A}$. These expressions are called *precedence relationships* of p. Set $\mathcal{R}$ can be empty.

(b) $\mathcal{S}$ is a set of expressions of the form $D[p_i] \equiv D[p_j]$, where p_i and p_j are PSPs in $\mathcal{P}$, and D is a dimension annotated in p_i and p_j. These expressions are called *node sharing expressions*. Set $\mathcal{S}$ can be empty.

(c) o is the name of one of the PSPs in $\mathcal{P}$. This PSP is called *output PSP* of the query. □

The term *structural expression* refers indiscreetly to a precedence relationship or to a node sharing expression.

Example 2.4 Consider the following query on $\mathcal{D}$: $Q = (\mathcal{P}, \mathcal{S}, p_2)$, where $\mathcal{P} = \{(p_1, \mathcal{A}_1, \mathcal{R}_1), (p_2, \mathcal{A}_2, \mathcal{R}_2)\}$,
$\mathcal{A}_1 = \{Location[p_1] = \text{ Athens}, \; Period[p_1] = \{\text{Day}\}\}$,
$\mathcal{R}_1 = \{\}$
$\mathcal{A}_2 = \{Format[p_2] = ?, \; Measure[p_2] = ?,$
$Period[p_2] = \{\text{Day}\}\}$,
$\mathcal{R}_2 = \{Period[p_2] \Rightarrow Format[p_2],$
$Period[p_2] \Rightarrow Measure[p_2]\}$, and
$\mathcal{S} = \{Period[p_1] \equiv Period[p_2]\}$. □

We graphically represent queries using graph notation. Consider a query Q. Each PSP of Q is represented as a (not necessarily connected) graph of dimensions labeled by their annotating expressions in the PSP. The name of the output PSP of Q is preceded by a $\star$. Child and descendent precedence relationships in a PSP are depicted using single ($\rightarrow$) and double ($\Rightarrow$) arrows between the respective nodes in the PSP graph. Two nodes (annotated dimensions) in different PSP graphs that participate in a node sharing expression of Q are linked in its graphical representation with a straight line labeled by the symbol '$\equiv$'. The query in Figure 4 is a graphical representation of query Q of Example 2.4.

Semantics. The answer of a query Q on a value tree T is a value tree, subtree of T. Every path from the root to a leaf of T is the image of the output path of Q under an embedding of Q into T that preserves the precedence relationships and nodes sharing expressions of Q. More formally:

Definition 2.4 Let T be a value tree over a dimension set $\mathcal{D}$, and Q be a query on $\mathcal{D}$. An *embedding* of Q into T is a mapping M of the annotated dimensions of the PSPs of Q to nodes in T such that:

(a) The annotated dimensions of a PSP in Q are mapped to nodes in T that are on the same path from the root of T.

(b) For every annotating expression $D[p_i] = V$ in Q, the label of $M(D[p_i])$ is a value in V, if V is a set, and it is a value of D, if V is a "?".

(c) For every precedence relationship $D_j[p] \rightarrow D_k[p]$ (resp. $D_j[p] \Rightarrow D_k[p]$) in Q, $M(D_k[p])$ is a child (resp. descendent) of $M(D_j[p])$ in T.

(d) For every node sharing expression $D[p_i] \equiv D[p_j]$ in Q, $M(D[p_i])$ and $M(D[p_j])$ coincide. □

Given an embedding M of a query Q into a value tree T, and a PSP p in Q, the path from the root of T that comprises all the images of the annotated dimensions of p under M and ends in one of them is called *image* of p under M and is denoted $M(p)$. Notice that more than one PSP of Q may have their image in the same root-to-leaf path of T (M does not have to be a bijection).

Definition 2.5 Let T be a value tree over a dimension set $\mathcal{D}$, and $Q = (\mathcal{P}, \mathcal{S}, o)$ be a query on $\mathcal{D}$. The *answer* of Q on T is a subtree T' of T such that:

(a) For every embedding of Q into T, the image of the output PSP of Q is in T'.

(b) Every root-to-leaf path of T' is the image of the output PSP of Q under an embedding of Q into T.

If there is no such a subtree T', the answer of Q on T is an empty tree. We say that the answer of Q on T is *empty*. □

Note that annotating a dimension with a "?" in a PSP of a query is different than omitting this dimension from the PSP.

Example 2.5 Consider the query Q of Example 2.4, graphically shown in Figure 4. Consider also the value tree T of Figure 1. The answer of Q on T is shown in Figure 5. There are two embeddings of Q into T which result in two distinct root-to-leaf paths in the answer of Q on T. Both embeddings map the dimensions of PSPs p_1 and p_2 of Q to nodes on the same path from the root of T. $\square$

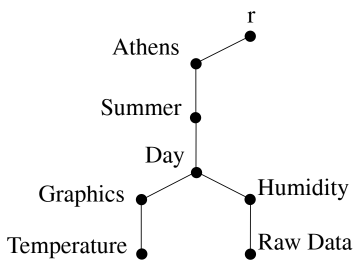

Figure 5. The answer of Q on T

3. Definitions and Related Concepts

We define in this section query containment and we provide additional concepts that will allow us to study the problem of checking two queries for containment.

3.1. Query Containment

We consider two types of query containment: query containment in the absence of a dimension graph (*absolute query containment*) and query containment with respect to a dimension graph (*relative query containment*).

Definition 3.1 Let Q_1 and Q_2 be two queries on $\mathcal{D}$. Query Q_1 *contains* query Q_2 (denoted $Q_2 \subseteq Q_1$) if and only if for every value tree T over $\mathcal{D}$, every root-to-leaf path in the answer of Q_2 on T is also a root-to-leaf path in the answer of Q_1 on T. Queries Q_1 and Q_2 on $\mathcal{D}$ are *equivalent* (denoted Q$_1 \equiv Q_2$) if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. $\square$

Example 3.1 Consider queries Q_1, Q_2, and Q_3 of Figures 6, 7 and 8 respectively. Queries Q_1 and Q_2 have three PSPs and query Q_3 has one PSP. It can be shown that $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_3$, while $Q_2 \not\subseteq Q_1$ and $Q_3 \not\subseteq Q_2$. $\square$

Queries are to be evaluated on value trees that underlie a specific dimension graph. Therefore, we also introduce the following 'relative' definition of query containment that takes into account the dimension graph under consideration.

Definition 3.2 Let Q_1 and Q_2 be two queries on $\mathcal{D}$, and $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. Query Q_1 *contains* query Q_2 *with respect to dimension graph* $\mathcal{G}$ (denoted $Q_2 \subseteq_{\mathcal{G}} Q_1$) if and only if for every value tree T over $\mathcal{D}$ underlying $\mathcal{G}$, every root-to-leaf path in the answer of Q_2 on T is also a root-to-leaf path in the answer of Q_1 on T. Queries Q_1 and Q_2 on $\mathcal{D}$ are *equivalent with respect to dimension graph* $\mathcal{G}$ (denoted $Q_2 \equiv_{\mathcal{G}} Q_1$) if and only if $Q_1 \subseteq_{\mathcal{G}} Q_2$ and $Q_2 \subseteq_{\mathcal{G}} Q_1$. $\square$

Clearly, given a dimension graph $\mathcal{G}$, if $Q_1 \subseteq Q_2$, then $Q_1 \subseteq_{\mathcal{G}} Q_2$. The opposite is not necessarily true.

Example 3.2 Consider the queries Q_1, Q_2 and Q_3 of Example 3.1, shown in Figures 6, 7, and 8 respectively. Consider also the dimension graph $\mathcal{G}$ of Figure 3. As we saw in Example 3.1, $Q_3 \not\subseteq Q_2$. However, as we show later, $Q_3 \subseteq_{\mathcal{G}} Q_2$. Since $Q_2 \subseteq Q_3$, it is also true that $Q_2 \subseteq_{\mathcal{G}} Q_3$. Therefore, $Q_2 \equiv_{\mathcal{G}} Q_3$. We show later that, in contrast, $Q_2 \not\subseteq_{\mathcal{G}} Q_1$. $\square$

3.2. Unsatisfiable Queries and Valid PSP Clusters

As with query containment, we can define query unsatisfiability in the presence of a dimension graph.

Definition 3.3 Let $\mathcal{G}$ be a dimension graph on D. A query on $\mathcal{D}$ is *unsatisfiable with respect to* $\mathcal{G}$ if its answer is empty on every value tree underlying $\mathcal{G}$. Otherwise, it is called *satisfiable with respect to* $\mathcal{G}$. $\square$

An unsatisfiable query is contained in any query with respect to a dimension graph. In [18], necessary and sufficient conditions for query unsatisfiability are provided. In the following, we assume that queries are satisfiable with respect to $\mathcal{G}$.

A set of PSPs in a query that are all linked together through node sharing expressions is called cluster:

Definition 3.4 A *cluster* is a set C of PSPs and node sharing expressions such that: for every partition of C in two non-empty sets there is a node sharing expression in Q involving a dimension other than R and PSPs from both sets. $\square$

Given a dimension graph $\mathcal{G}$, it is possible that there is a cluster that can be added to any query without affecting its answer on any value tree that underlies $\mathcal{G}$. To deal with this issue, we introduce the concept of valid cluster.

Definition 3.5 Let $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. A cluster Q is *valid* with respect to $\mathcal{G}$ if and only if, for every value tree T over $\mathcal{D}$ underlying $\mathcal{G}$, there is a mapping M of the annotated dimensions of p to nodes of T that satisfies the conditions (a), (b), (c) and (d) of definition 2.4 (i.e M is an embedding of Q into T). $\square$

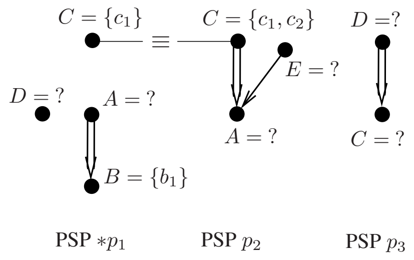

Figure 6. Query Q_1

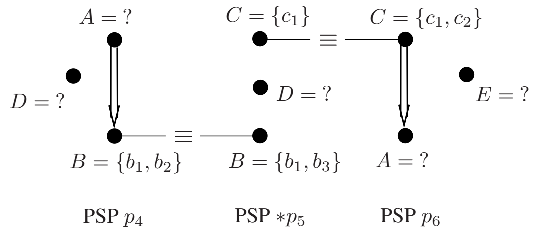

Figure 7. Query Q_2

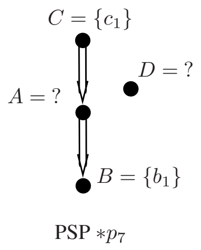

Figure 8. Query Q_3

Clearly, adding to a query Q a valid cluster or removing from a query a valid cluster that does not include the output PSP of Q and does not share nodes with PSPs outside the cluster results in a query equivalent to Q w.r.t. $\mathcal{G}$.

In the following, we assume that a query does not comprise a valid disconnected cluster that does not contain the output PSP of the query.

A $\equiv$-swing in a query is a set of two node sharing expressions involving three paths such that no order can be induced between them in the query:

Definition 3.6 An $\equiv$-swing in a query is a set of two node sharing expressions $A[p_1] \equiv A[p_2]$ and $B[p_2] \equiv B[p_3]$ such that A and B are distinct dimensions and no precedence relationship can be inferred between A and B in p_2 from the expressions in Q. $\square$

$\equiv$-swings in a query Q can create discrepancies because they may force every tree in which there is an embedding of a query Q to comprise a path that involves a number of dimensions which together do not appear in any PSP of Q. In the following, we assume that if query has an $\equiv$-swing $A[p_1] \equiv A[p_2]$ and $B[p_2] \equiv B[p_3]$, dimensions A and B appear in both PSPs p_1 and p_3.

3.3. Structural Expression Inference and Query Full Form

Because tree patterns are partially specified in the queries, new, non-trivial structural expressions (precedence relationships and node sharing expressions) can be inferred from the expressions specified in the queries. These structural expressions are preserved by all the embeddings of the query to a value tree; in other words, adding these structural expressions to the query does not remove paths from its answer on any value tree. We formalize below the notion of structural expression implication.

Definition 3.7 Let $\mathcal{E}$ be a set of expressions of a query Q on a dimension set $\mathcal{D}$, and e be a structural expression that involves PSP names and dimensions in $\mathcal{E}$. We say that e is *implied* from $\mathcal{E}$ if and only if for every value tree T over $\mathcal{D}$ and every embedding M of Q into T, M preserves e. $\square$

Example 3.3 Consider the set of expressions $\mathcal{E} = \{A[p_1] \Rightarrow B[p_1], A[p_1] \equiv A[p_2], R[p_2] \Rightarrow B[p_2]\}$. One can see that $\mathcal{E}$ implies the precedence relationship $A[p_2] \Rightarrow B[p_2]$. $\square$

The *closure* of a set $\mathcal{E}$ of expressions is the set that includes the expressions in $\mathcal{E}$ and those structural expressions that can be implied from $\mathcal{E}$. A set of inference rules for structural expression implication has been provided in [18]. These inference rules allow us to compute efficiently closures of sets of expressions. In order to check queries for containment, we introduce below the concept of the full form of a query which is based on the concept of closure of a set of expressions.

Definition 3.8 Let Q be a query and $\mathcal{E}$ be its set of expressions. Let $\mathcal{N} = \{R[p] \Rightarrow D[p] \mid D$ is an annotated dimension in the PSP p of Q, and $D \neq R\}$. Let $\mathcal{E}' = \mathcal{E} \cup \mathcal{N}$. That is, $\mathcal{E}'$ is the set of expressions of Q that includes also ancestor precedence relationships from dimension R to every annotated dimension in every PSP of Q. Query Q is in *full form* if:

(a) $\mathcal{E}$ is equal to the closure of $\mathcal{E}'$, and
(b) For every node sharing expression $D[p_1] \equiv D[p_2]$ in the closure of $\mathcal{E}'$, the annotations of dimension D in PSPs p_1 and p_2 of Q are the same.

A query Q' is the *full form* of query Q if and only if Q' is in full form and is equivalent to Q. $\square$

The full form of a query Q can be computed by: (a) replacing in Q its set of structural expressions $\mathcal{E}$ by the closure of $\mathcal{E}'$, (b) annotating by a '?' every dimension $D[p]$ in $\mathcal{E}'$ that is not annotated in Q, and (c) annotating every dimension D in PSPs p_1 and p_2 by the intersection of its annotations in PSPs p_1 and p_2 in Q, if the node sharing expression $D[p_1] \equiv D[p_2]$ is in $\mathcal{E}'$ (recall that a '?' as an annotation denotes the set of all the values of the corresponding dimension). Clearly, the above procedure constructs a query that is equivalent to Q.

To graphically represent queries in full form, by convention, we do not depict (a) double arrows (ancestor precedence relationships) from R, (b) double arrows that can be transitively derived from other double arrows in the same

PSP, and (c) double arrows that can be directly derived form single arrows in the same PSP. All the omitted double arrows can be easily derived from the (single and/or double) arrows that are explicitly shown in the query graph.

Example 3.4 Consider the queries Q_1 and Q_2 of Figures 6 and 7. Figures 9 and 10 show a full form of Q_1 and Q_2, respectively. For instance, the full form of query Q_1 can be generated using the following inference rules instances: $A[p_1] \Rightarrow B[p_1] \ \vdash \ R[p_1] \Rightarrow A[p_1]$,
$C[p_2] \Rightarrow A[p_2],\ E[p_2] \rightarrow A[p_2] \ \vdash \ C[p_2] \Rightarrow E[p_2]$, and
$C[p_2] \ \rightarrow \ C[p_2],\ C[p_2] \ \equiv \ C[p_1], R[p_1] \ \rightarrow \ A[p_1] \ \ \vdash$ $C[p_1] \Rightarrow A[p_1]$. Query Q_3 of Figure 8 is in full form. $\square$

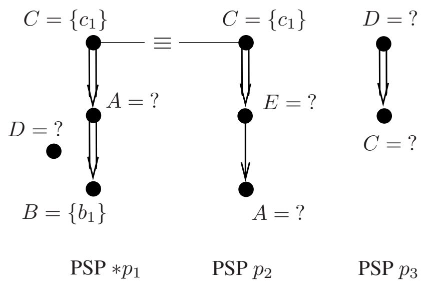

Figure 9. A full form of Query Q_1

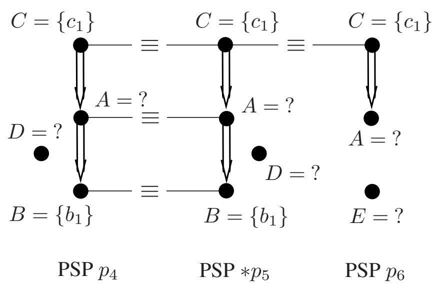

Figure 10. A full form of Query Q_2

4. Checking Query Containment

We study below absolute query containment, and relative query containment.

4.1. Absolute Query Containment

In order to address absolute query containment we need the concept of homomorphism between partially specified tree pattern queries:

Definition 4.1 Let Q_1 and Q_2 be two queries on $\mathcal{D}$. An *homomorphism* from Q_2 to Q_1 is a mapping m from the annotated dimensions of Q_2 to the annotated dimensions of Q_1 such that:

(a) If the annotated dimension n is labeled by dimension D in Q_2, then $m(n)$ is also labeled by D in Q_1. All the annotated dimensions of a PSP in Q_2 are mapped under m to annotated dimensions in the same PSP of Q_1.
(b) If an annotated dimension n in Q_2 is annotated by $V_2 \neq$?, then $m(n)$ in Q_1 is annotated by V_1 such that $V_1 \subseteq V_2$.
(c) The annotated dimensions in the output PSP o_2 of Q_2 are mapped under m to annotated dimensions in the output PSP o_1 of Q_1, and every annotated dimension in o_1 is the image under m of an annotated dimension in o_2.
(d) If $D[p] \rightarrow D'[p]$ (resp. $D[p] \Rightarrow D'[p]$) is in Q_2, then $m(D[p]) \rightarrow m(D'[p)$ (resp. $m(D[p]) \Rightarrow m(D'[p])$) is in Q_1.
(e) If $D[p] \equiv D[p']$ is in Q_2, then $m(D[p])$ and $m(D[p'])$ coincide or $m(D[p]) \equiv m(D[p'])$ is in Q_1. $\square$

The next proposition provides necessary and sufficient conditions for absolute query containment in terms of homomorphisms between partially specified tree pattern queries.

Proposition 4.1 Let Q_1 and Q_2 be two queries on $\mathcal{D}$ in full form. $Q_1 \subseteq Q_2$ if and only if there is a homomorphism from Q_2 to Q_1. $\square$

Sketch: Sufficiency is not difficult. To show necessity we generate a "minimal" tree T on which there is an embedding of Q_1. Since $Q_1 \subseteq Q_2$, Q_2 also has an embedding to T. We use those two embeddings to generate a mapping from Q_2 to Q_1. $\square$

Example 4.1 Consider the queries Q_1, Q_2 and Q_3 of Example 3.1 shown in Figures 6, 7, and 8. Their full form is shown in Figures 9, 10, and 8, respectively. Proposition 4.1 proves the claim of Example 3.1 that $Q_1 \subseteq Q_2$ through the homomorphism from Q_2 to Q_1 induced by the following mapping on PSPs: $p_4 \rightarrow p_1$, $p_5 \rightarrow p_1$, and $p_6 \rightarrow p_2$. Similarly, the homomorphism from Q_3 to Q_2 induced by the mappings $p_7 \rightarrow p_5$ proves that $Q_2 \subseteq Q_3$.

The absence of homomorphism from Q_1 to Q_2 and from Q_2 to Q_3 proves that $Q_2 \not\subseteq Q_1$, and $Q_3 \not\subseteq Q_2$. $\square$

4.2. Relative Query Containment

In order to address relative query containment, we need the concept of dimension tree: given a query Q and a dimension graph $\mathcal{G}$, a *dimension tree* of Q on $\mathcal{G}$ is a tree U whose nodes are labeled by annotated dimensions and its root is labeled by the annotated dimension R. It also has a distinguished node called *output node* that defines a path from its root called *output path.* A dimension tree U of Q on $\mathcal{G}$ satisfies two properties. First, there is a mapping m from the nodes of U to the nodes of $\mathcal{G}$ that is root-preserving, and respects labeling dimensions and child relationships. Second, there is a mapping m' from the annotated nodes of Q to the nodes of U that respects labeling dimensions, annotations (existentially), precedence relationships, and node sharing expressions. Intuitively, a dimension tree for Q on $\mathcal{G}$ represents an embedding of Q into $\mathcal{G}$ that respects labeling dimensions, precedence relationships, node sharing expressions and annotations (existentially). This embedding is the composition of m' and m. The dimensions trees of Q on $\mathcal{G}$ represent all such possible embeddings of Q into $\mathcal{G}$.

Example 4.2 Figure 11 shows the dimension trees of query

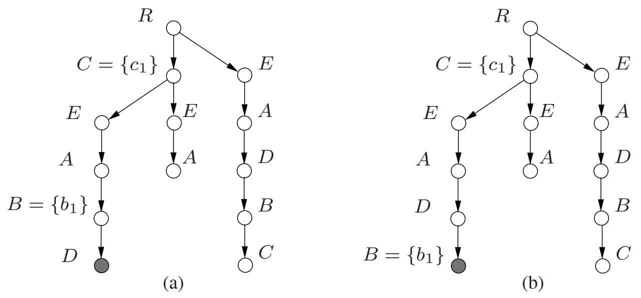

Figure 11. The dimension trees of query Q_1 on dimension graph $\mathcal{G}$: (a) U_1^1, (b) U_1^2

Q_1 of Figure 6 on the dimension graph $\mathcal{G}$ of Figure 3. Dimension annotations that are '?' are not shown in the graphical representation of dimension trees. □

A dimension tree U of a query Q on a dimension graph $\mathcal{G}$ can be seen as a (structurally) completely specified query: root-to-leaf paths determine PSPs; the output path determines the output PSP; edges determine child precedence relationships; common nodes of two paths determine node sharing expressions. Such queries are (structurally) completely specified because they form a tree pattern (without missing edges) involving only parent-child (and not ancestor-descendant) relationships. Since dimension trees are also queries, we can apply to them the concepts defined on queries.

Given a dimension graph $\mathcal{G}$, a query Q represents a set of completely specified queries (dimension trees). It can be shown that every path in the answer of Q on a value tree T underlying $\mathcal{G}$ is also a path in the answer of some of the dimension trees of Q, and conversely. Therefore, the answer of Q on T can be constructed by merging the answers of the dimension trees of Q on T.

We now state a proposition that provides necessary and sufficient conditions for relative query containment, in terms of absolute dimension tree containment.

Proposition 4.2 Let Q_1 and Q_2 be two queries on $\mathcal{D}$ and $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. Let also $\mathcal{U}_1$ be the set of dimension trees of Q_1 on $\mathcal{G}$ and $\mathcal{U}_2$ be the set of dimension trees of Q_2 on $\mathcal{G}$. $Q_1 \subseteq_{\mathcal{G}} Q_2$ if and only if there is a mapping m from $\mathcal{U}_1$ to $\mathcal{U}_2$ such that, for every dimension tree U in $\mathcal{U}_1$, $U \subseteq m(U)$. □

Example 4.3 Consider the queries Q_1, Q_2 and Q_3 of Example 4.2, shown in Figures 6, 7, and 8, and the dimension graph $\mathcal{G}$ of Figure 3. Figures 11, 12, and 13 show the dimension trees of Q_1, Q_2, and Q_3 on $\mathcal{G}$, respectively.

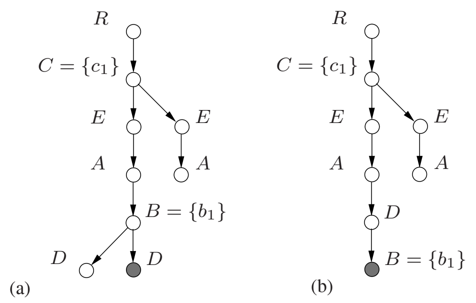

Figure 12. The dimension trees of Q_2 on $\mathcal{G}$: (a) U_2^1, (b) U_2^2

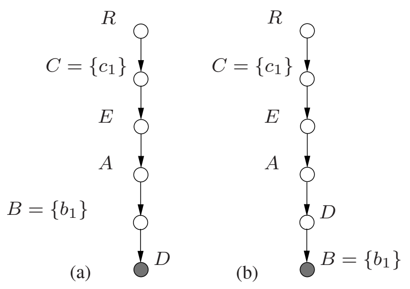

Figure 13. The dimension trees of Q_3 on $\mathcal{G}$: (a) U_3^1, (b) U_3^2

Proposition 4.2 proves the claim of Example 4.2 that $Q_3 \subseteq_{\mathcal{G}} Q_2$: let m be the mapping $m(U_3^1) = U_2^1$ and $m(U_3^2) = U_2^2$. Based on Proposition 4.1 we can show that $U_3^1 \subseteq U_2^1$ and $U_3^2 \subseteq U_2^2$. Therefore, $Q_3 \subseteq_{\mathcal{G}} Q_2$. Proposition 4.2 also proves that, in contrast, $Q_2 \not\subseteq_{\mathcal{G}} Q_1$. □

5. Experimental Evaluation

We implemented and experimentally evaluated our approach on a prototype system. Checking relative query containment can be time consuming since, as we saw in the previous section, it involves checking for absolute containment pairs of dimension trees whose number may be substantial. In contrast, checking absolute query containment requires only the detection of an homomorphism between the two queries. For this reason, we suggest a technique for checking query containment with respect to a dimension graph

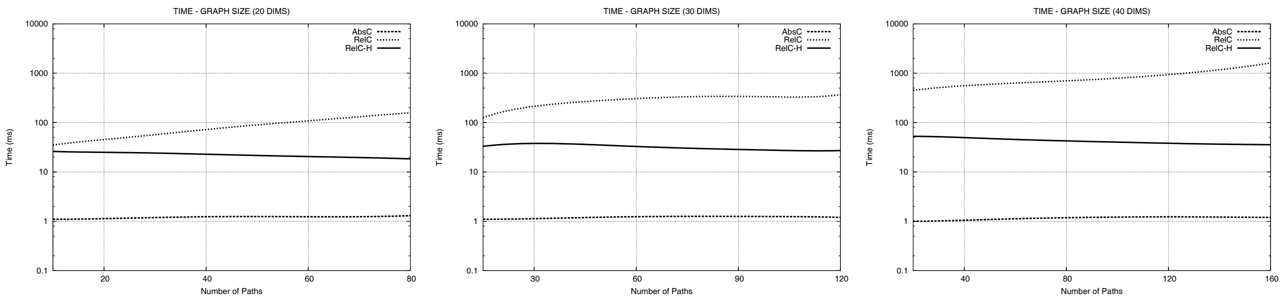

Figure 14. Execution time for checking query containment varying the number of root-to-leaf paths for different numbers of dimensions in the dimension graph.

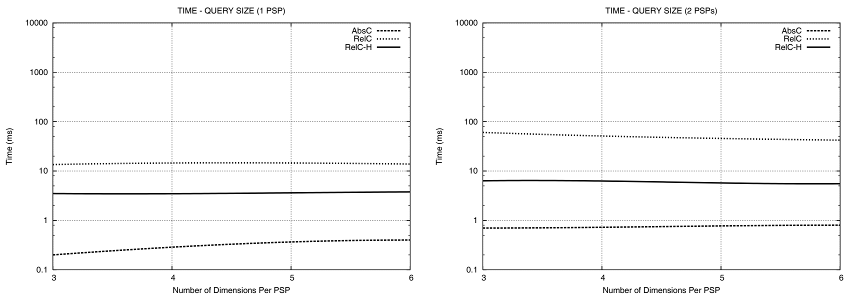

Figure 15. Execution time for checking query containment varying the number of dimensions per PSP for different numbers of PSPs in the query.

$\mathcal{G}$ that reduces relative query containment to absolute query containment.

The basic idea of our technique is the following. Suppose that we want to check whether $Q_1 \subseteq_{\mathcal{G}} Q_2$. For every precedence relationship $X \Rightarrow Y$ (or $X \rightarrow Y$), we extract, in advance, from $\mathcal{G}$ all the precedence relationships that hold on all the paths from the root of $\mathcal{G}$ that satisfy $X \Rightarrow Y$. For instance, in the dimension graph of Figure 3, for the precedence relationship $A \Rightarrow B$, we can extract the precedence relationships $R \Rightarrow E, R \Rightarrow A, R \Rightarrow B, E \Rightarrow A, E \Rightarrow B, A \Rightarrow B$, and $E \rightarrow A$. Clearly, if a PSP in Q_1 involves $X \Rightarrow Y$, the extracted precedence relationships can be added to it to form an equivalent query Q_1'. Then, Q_1' is checked for *absolute* containment into Q_2. This approach is sound but not complete.

We used a set of queries to be evaluated on value trees that underlie dimension graphs, and we measured the execution time for checking query containment in three different cases: (a) absolute query containment, (b) relative query containment, (c) relative query containment using our technique. We also measured the accuracy of our technique with respect to (b). For every measure point, 100 pairs of queries were generated (10 pairs of queries for each one of 10 dimension graphs generated). The recorded execution time per point is the average execution time. It does not include the time required to extract the precedence relationships from the dimension graph which are assumed to be pre-computed. In the experiments, we used dimension graphs whose number of root-to-leaf paths does not exceed twice the number of their nodes. This number is selected to comply with the dimension graphs we extracted from several popular XML benchmarks (e.g. XMark, XMach)[1].

Execution time and accuracy varying the density of the dimension graph. We measured the execution time and the accuracy for checking query containment varying the number of root-to-leaf paths for different numbers of dimensions in the dimension graph. In Figure 14, we present the results obtained for graphs having 20, 30 and 40 dimensions, regarding the execution time. The number of PSPs in the queries is fixed to 3.

As expected, checking relative query containment is clearly time consuming compared to absolute query containment. In all cases, our technique clearly improves the checking of relative query containment.

Regarding the accuracy, for a low number of root-to-leaf paths in the dimension graph, the percentage of correct answers of our technique (accuracy) is over 80%. As the number of paths increases, the percentage drops. This drop is sharper for higher numbers of dimensions. In all cases the percentage is above 50%.

Execution time and accuracy varying the density of

[1]http://monetdb.cwi.nl/xml/,
http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html

queries. We measured the execution time and the accuracy for checking query containment varying the number of dimensions per PSP for different numbers of PSPs in the query. In Figure 15, we present the results obtained for queries having 1 and 2 PSPs, concerning the execution time. The number of dimensions and the number of paths in the dimension graph are fixed to 30 and 15 respectively.

Again, our technique clearly improves the checking of relative query containment. The improvement becomes more important as the number of PSPs involved in a query increases.

Regarding the accuracy, the percentage of correct query containment checks is in all cases above 93%.

6. Conclusion

We addressed the containment problem for partially specified tree-pattern queries. A central feature of this type of queries is that the structure can be specified fully, partially, or not at all in a query, and thus can be used for querying of data sources when their structure is not fully known to the user. To support the evaluation of partially specified queries we used semantically rich constructs, called dimension graphs, which abstract structural information of the tree-structured data. We studied the problem of query containment in the absence (absolute query containment) and in the presence (relative query containment) of dimension graphs and we provide necessary and sufficient conditions for each type of query containment. Also, we suggested a technique for relative query containment based on structural information extracted from the dimension graph. We validated our technique experimentally, showing that it clearly improves checking of relative query containment.

Our future work will focus on suggesting a sequence of heuristics for extracting structural information from the dimension graph, and studying how they gradually trade precedence relationship extraction time for accuracy in checking relative query containment.

References

[1] XML Path Language (XPath). World Wide Web Consortium site, W3C XPath: http://www.w3.org/TR/xpath20.

[2] XML Query (XQuery). World Wide Web Consortium site, W3C XQuery: http://www.w3.org/XML/Query.

[3] XSL Transformations (XSLT). World Wide Web (W3C) Consortium site, http://www.w3.org/TR/xslt.

[4] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 497–508, 2001.

[5] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *Proc. of the 8th Intl. Conf. on Extending Database Technology, Prague, Czech Republic*, 2002.

[6] D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM Symp. on Principles of Database Systems*, pages 149–158, 1998.

[7] S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale xml repository. In *Proc. of the 27th Intl. Conf. on Very Large Data Bases*, 2001.

[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A Semantic Search Engine for XML. In *Proc. of the 29th Intl. Conf. on Very Large Data Bases*, 2003.

[9] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data, Madison, USA*, pages 287–298, 2002.

[10] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. In *Proc. of the 19th Intl. Conf. on Data Engineering*, pages 367–378, 2003.

[11] L. V. S. Lakshmanan, G. Ramesh, H. W. Wang, and Z. J. Zhao. On Testing Satisfiability of Tree Pattern Queries. In *Proc. of the 30th Intl. Conf. on Very Large Data Bases*, pages 120–130, 2004.

[12] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free Xquery. In *Proc. of the 30th Intl. Conf. on Very Large Data Bases*, pages 72–83, 2004.

[13] G. Miklau and D. Suciu. Containment and Equivalence for an XPath Fragment. In *Proc. of the 21st ACM Symp. on Principles of Database Systems*, pages 65–76, 2002.

[14] F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *Proc. of the 13th Intl. Conf. on Database Theory*, pages 315–329, 2003.

[15] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *SIGMOD Conference*, pages 455–466, 1999.

[16] P. Ramanan. Efficient Algorithms for Minimizing Tree Pattern Queries. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data, Madison, USA*, pages 299–309, 2002.

[17] D. Theodoratos and T. Dalamagas. Querying Tree Structured Data Using Dimension Graphs. In *Proc. of the 17th Intl. Conf. on Advanced Information Systems Engineering*, pages 201–215, 2005.

[18] D. Theodoratos, T. Dalamagas, A. Koufopoulos, and N. Gehani. Semantic Querying of Tree-Structured Data Sources Using Partially Specified Tree-Patterns. In *Proc. of the 14th ACM International Conference on Information and Knowledge Management*, pages 712–719, 2005.

[19] P. T. Wood. Minimising Simple XPath Expressions. In *Informal Proc. of the 4th Intl. Workshop on the Web and Databases*, pages 13–18, 2001.

[20] P. T. Wood. Containment for XPath Fragments under DTD Constraints. In *Proc. of the 13th Intl. Conf. on Database Theory*, pages 300–314, 2003.