Assigning semantics to partial tree-pattern queries

Dimitri Theodoratos *, Xiaoying Wu

Department of Computer Science, New Jersey Institute of Technology, USA

Received 12 October 2006; accepted 24 July 2007
Available online 7 August 2007

Abstract

The wide adoption of XML has increased the interest on data models that are based on tree-structured data. Querying capabilities are provided through tree-pattern queries (TPQs). The need for querying tree-structured data sources when their structure is not fully known, and the need to integrate multiple data sources with different tree structures have driven, recently, the suggestion of query languages that relax the complete specification of a tree pattern. Assigning semantics to the queries of these languages so that they return meaningful answers is a challenging issue.

In this paper, we introduce a query language which allows the specification of partial tree-pattern queries (PTPQs). The structure in a PTPQ can be flexibly specified fully, partially or not at all. We define index graphs which summarize the structural information of data trees. Using index graphs, we show that PTPQs can be evaluated through the generation of an equivalent set of “complete” TPQs. We suggest an original approach that exploits the set of complete TPQs of a PTPQ to assign meaningful semantics to the PTPQ language. In contrast to previous approaches that operate locally on the data to compute meaningful answers (usually by computing lowest common ancestors), our approach operates globally on index graphs to detect meaningful complete TPQs. We implemented and experimentally evaluated our approach on DBLP-based data sets with irregularities. Its comparison to previous ones shows that it succeeds in finding all the meaningful answers when the others fail (perfect recall). Further, it outperforms approaches with similar recall in excluding meaningless answers (better precision). Finally, it is superior to and scales better than the only previous approach that allows for structural constraints in the queries. Our approach generates TPQs and therefore, it can be easily implemented on top of an XQuery engine.

© 2007 Elsevier B.V. All rights reserved.

Keywords: XML; Partial tree-pattern query; Keyword query; Query language semantics; Meaningful answer; Structural summary of XML data

1. Introduction

XML is by now the de facto standard for exporting and exchanging data on the web. In this model, data is represented in a tree structured form.1 Query languages for XML are mainly based on the specification of tree

* Corresponding author.
E-mail addresses: dth@njit.edu (D. Theodoratos), xw43@njit.edu (X. Wu).
1 ID/IDREF links would require the modeling of XML documents as graphs but we ignore these features here for simplicity.

1.1. The problem

XML data does not have to comply with a schema. Even if the data comply with some schema, a user might not have full knowledge of it. Formulating a TPQ that will retrieve the desired results becomes complex. The reason is that, as we explain below, the user has to specify an order for the elements in every path of a tree-pattern query (TPQ) even though (a) the user might not know this order, and (b) the user might not be interested in imposing an order as a structural restriction in the query. Further, data sources usually export data on the web under different structures even if they export the same information or information from the same knowledge domain. Since elements may be ordered differently in these structures, querying all these data sources in an integrated way becomes an issue: usually, a single TPQ is not able to retrieve the desired information from all of them.

Consider, for instance, the TPQs shown in Fig. 1 for a bibliographic XML document. These queries involve descendant relationships (depicted with double arrows) which denote a path of zero or more elements between two elements, and wildcards (labeled by a '*') which indicate the presence of some element in a path without specifying which one. The output node (the node returned to the user) is shown boxed.

Even though TPQs provide some freedom in the specification of a tree structure (e.g. through the use of descendant relationships and wildcards), they all have a common restrictive structural requirement: in every root-to-leaf path, there is a total order for the nodes. It is not possible in a TPQ to indicate that two nodes \( n_1 \) and \( n_2 \) occur in a path without specifying a precedence relationship between them: node \( n_1 \) has to precede node \( n_2 \) or vice-versa.

Specifying queries in this context with languages like XPath or XQuery is a complex task [9,14,27,22]. In addition, the user might have to specify a number of tree patterns that are exponential on the number of elements in the query [27,22]. To deal with this problem several approaches try to introduce query languages that relax the structure of a tree pattern in a query [24,14,9,19,20]. An even more challenging issue is to assign semantics to these queries so that they return to the user fragments of the XML tree that are meaningful (that is, the queries are not matched to unrelated parts of the XML document).

1.2. Limitations of previous approaches

Two different solutions have been suggested to deal with the dependence of queries for XML from the structure of the data: one that adapts the structureless keyword-search techniques to query tree-structured data, and one that extends structured TPQ languages with keyword search capabilities.

The first solution modifies keyword-based techniques used by search engines for HTML to distinguish between text (data) and elements (metadata). It also modifies these techniques to return fragments of the documents that contain the keywords, as is appropriate for XML, instead of links to documents [24,14,9]. This solution offers a very convenient way for specifying queries, even for a naive user. Nevertheless, its major limitation is that structural restrictions cannot be specified in the query. Structural restrictions are necessary when querying tree-structured data for two reasons: (a) they can express user requirements and therefore, refine the query answer, and (b) they can express structural constraints that are known to hold in order to speed up the evaluation of the queries.
The second solution is applied to extend structured query languages for XML to enable keyword search [10,19,20]. However, these languages cannot avoid having a syntax which is complex for the simple user [9,14,27,22].

To face the challenge of assigning semantics to queries so that they return meaningful answers, most approaches exploit directly or indirectly the notion of Lowest Common Ancestor (LCA) of a set of nodes in the XML tree. However, in most practical cases, the information in the XML tree is incomplete (e.g. optional elements/values in the schema of the document are missing), or irregular (e.g. different structural patterns coexist in the same document) [3]. For instance, examining the DBLP data set (data collected in May 2006) we found that almost 10% of the “book” entries and over 1% of “article” entries do not have an author, while almost all “proceedings” entries do not have authors (this latter one is reasonable and expected). In such cases, the approaches of the first solution (the structureless keyword-based solution), even if they succeed in retrieving all the meaningful answers, they comprise only a tiny percentage of meaningful answers in their answer set. Most of the answers are meaningless. In other words, these approaches have low precision. Our experiments in Section 7 with DBLP-based data sets show that in some cases the precision falls below 1% for some approaches. Clearly, such a low precision is a serious limitation for those approaches.

A recent approach (MLCAS [19,20]) of the second solution (the solution that extends TPQs with keyword capabilities) shows improved precision. However, the percentage of meaningful answers returned (i.e. the recall) is low. In our experiments in Section 7, the recall of the MLCAS approach falls below 60% for several cases of XML data. Clearly, the poor recall cannot be improved by further imposing structural restrictions. This performance is not satisfactory for data integration environments for which this approach is intended. In addition, it employees different semantics for the keyword part (MLCAS) and the structured part of a query (XQuery). As a consequence, structural restrictions in a query cannot be used to recuperate answers that are not returned by the keyword search.

1.3. Our approach

In this paper, we introduce an original approach which addresses the problem of query dependence from the structure by allowing a partial specification of tree patterns in queries. Partial tree-pattern queries (PTPQs) relax the requirement of total node order in the paths of the tree pattern. Structural restrictions in PTPQs can be flexibly specified fully, partially or not at all. To provide some intuition, we show in Fig. 2 a simple PTPQ.

This PTPQ has three (partial) paths surrounded by dotted lines. Nodes are labeled. Node labels preceded by a * sign denote variables (similar to wildcards in XPath but named). The nodes in a partial path are not necessarily related through a total order: an author can precede a title in a path or vice versa. Undirected edges labeled by the symbol / between two nodes from different partial paths indicate that these two nodes coincide (that is, these two nodes denote a single node shared by the two partial paths). For instance, in Fig. 2, the two nodes labeled by “subject” denote a common node of partial paths 1 and 2. Notice that PTPQs can be structureless keyword queries when they comprise a single element in each partial path. They can also be queries with partial specification of the tree structure. Such an example is the query of Fig. 2. Finally, they can be queries that define regular tree patterns like those of Fig. 1.

We suggest an original approach for assigning semantics to our PTPQ language. The originality of our approach relies on the use of structural summaries of the XML document for identifying structural patterns (TPQs) for a given query. Using a partial order between TPQs, we identify those of them (called meaningful
TPQs) that return meaningful answers. Previous approaches identify meaningful answers by operating \textit{locally} on the \textit{data} (usually computing Lowest Common Ancestors of nodes in the XML tree). In contrast, our approach operates \textit{globally} on \textit{structural summaries} of data to compute meaningful TPQs. This overview of data gives an advantage to our approach compared to previous ones.

1.4. Contribution

The main contributions of this paper are the following:

- We introduce a query language with variables (wildcards) that allows partial specification of the structure of the tree patterns. Queries in this language do not cast to the structure of a specific XML tree. The PTPQ language can be used by a user to specify structural constraints on keywords, without the knowledge of a complex TPQ language like XQuery.
- We define structural summaries of data called index graphs. We show how index graphs can be used to compute a set of fully specified TPQs (complete TPQs) for a PTPQ that together compute the answers of the PTPQ.
- We use a partial order on the complete TPQs for a PTPQ to determine those of them that are meaningful. The meaningful TPQs are used to assign semantics to PTPQs. The novelty of our approach relies on the fact that not only the data, but also tree patterns in the structure of the XML tree are taken into account in specifying the meaningful answers of a PTPQ. Further, our approach seamlessly applies this semantics both to PTPQs without structural restrictions (keyword queries) and to PTPQs that fully specify tree patterns.
- We compare our approach with previous approaches that query tree structured data with, and without, structural restrictions. We analyse cases where our approach succeeds in returning meaningful answers that escape other approaches. We also analyse cases where our approach succeeds in excluding meaningless answers that are returned by other approaches.
- We have implemented and experimentally evaluated our approach. Our approach shows better recall compared to previous ones. In addition, in most cases, it allows for a better precision among approaches with similar recall. The precision can be further improved since our approach allows for specifying structural restrictions the user might want to impose or knows that they hold on the data. This feature is not available to languages that are restricted to keyword queries.
- Besides the quality of the results, we have evaluated our approach in terms of performance. Our experimental results show that it largely outperforms and scales better than the only previous approach that allows for keywords and structured queries.
- The meaningful TPQs generated by our approach are tree patterns. Therefore, they can be evaluated using any XQuery engine, and their execution can benefit from optimization techniques developed up to now for XQuery (e.g. [15,4,7]).

1.5. Outline

The rest of the paper is organized as follows. The next section discusses related work. Section 3 presents the data model and the query language. Index graphs and complete TPQs for a PTPQ are introduced in Section 4. In Section 5, we present our novel semantics for the PTPQ language. Our approach is compared with previous one in Section 6. It is experimentally evaluated in terms of quality of results and performance in Section 7. We conclude and suggest future work in Section 8.

2. Related work

A traditional data integration approach would use mapping rules between a global structure and local structures [8] to generate multiple TPQs. However, this approach requires extensive manual intervention and is complex and subject to errors.
Olteanu et al. [23] and Olteanu [22] consider XPath expressions with backward predicates and show that they can be equivalently rewritten as sets of expressions with forward predicates. These works relate to ours since we also evaluate PTPQs by generating equivalent sets of TPQs. However, they differ from ours in two respects: (a) they do not address the issue of assigning meaningful semantics to queries which is the main focus of our paper, and (b) in rewriting queries, they do not consider index graphs. The use of index graphs allows us not only to limit the number of tree-pattern queries generated to those that reflect patterns existing in the data, but also to generate complete TPQs that is, TPQs that do not involve ancestor–descendant structural relationships and wildcards.

A number of papers deal with the assignment of meaningful semantics to keyword-based query languages for XML [24,14,9,19,27,20]. All of them are based on some variation of the concept of Lowest Common Ancestor (LCA). The query language in [9] allows also some primitive structural restrictions to be expressed. Li et al. [19,20] provides an extension of XQuery to allow users to query an XML document without full knowledge of the structure. It uses the concept of Meaningful Lowest Common Ancestor Structure (MLCAS) of a set of nodes for assigning semantics to keyword queries. We analytically compare our approach with the three approaches in [24,9,19,20] in Section 6 and experimentally in Section 7. Our approach shows better recall in all cases, including cases where the XML data are incomplete. Among approaches with similar recall, our approach shows better precision in most cases. In [27] the concept of Smallest Lowest Common Ancestor (SLCA) is used to assign semantics to keyword queries. This semantics is similar to that of the MLCA approach. Therefore, we do not directly compare it to ours.

In order to cope with low precision some approaches extend the database techniques with information retrieval techniques. In this direction, they rank the answers of keyword search queries on XML documents according to their estimated relevance [9,12]. Information retrieval systems using ranking functions may trade recall for precision. The PTPQ language is a database query language. Therefore, it does not employ any ranking functions. Its goal is to not miss any meaningful answer and to exclude as many meaningless answers as possible.

Some languages employ approximation techniques to search for answers when the initial query is too restricted to return any. They either relax the structure of the queries or the matchings of the queries to the data [16,5]. In contrast to our language, these languages return approximate (not exact) answers.

Several papers focus on providing efficient algorithms for evaluating LCAs for keyword queries [24,14,9,19,20,27,13]. Our approach is different and does not have to explicitly compute LCAs of nodes in the XML tree. In contrast, it computes a number of meaningful TPQs for PTPQs that involve keywords and/or structural restrictions. This computation is done mainly on a structural summary of data, which is usually much smaller than the data itself. For this reason, our experimental results show that our approach largely outperforms the MLCAS approach. Since TPQs can be evaluated using an XQuery engine, our approach can directly take advantage of the various optimization techniques developed so far for XQuery [15,4,7].

A partial tree-pattern query language is introduced in [25] and its query containment problem is addressed in [26]. However, this language does not comprise variables (wildcards) and descendant-or-self precedence relationships as the language we consider in this paper. More importantly, the problem of assigning meaningful semantics to the queries is not addressed in those papers.

3. The partial tree-pattern query language

We present in this section the data model and our query language that allows for partially specifying the structure of a tree pattern. Our goal in this paper is to focus on the structural aspects of the query language. Therefore, we abstract other features that a full-fledged commercial query language would normally comprise.

3.1. Data model

Let $\mathcal{E}$ be an infinite set of elements that includes a distinguished element $r$, $\mathcal{X}$ be an infinite set of variables, and $\mathcal{V}$ be an infinite set of values. Variables range over elements, and play the role of wildcards in tree-pattern queries. Here, we use variables to distinguish between different wildcard nodes. Symbols $e$, $x$, and $v$ (possibly with indices) refer systematically to an element, a variable, and a value respectively. The term construct (denoted $c$) refers indiscreetly to an element or a variable.
As is usual, we model XML documents as trees. Nodes in an XML tree are labeled by elements or values. In particular, the root node of an XML tree is the only node labeled by element \( r \). Values can label only leaf nodes. Attributes of elements in an XML document are modeled as (sub)elements. For simplicity we assume that the same element does not label two nodes on the same path (that is, the XML trees are not recursive). We discuss in the next section how this restriction can be relaxed.

Fig. 3 shows three XML trees \( T_1, T_2, \) and \( T_3 \) from different data sources that record bibliographic information in different formats (a slight extension of an example introduced in [19,20]). \( T_1 \) and \( T_3 \) categorize the data based on the publication year, while \( T_2 \) categorizes the data based on the type of publication (article or book). Still, in \( T_1 \) the year of the publications is specified as a child element of a “bib” node, while in \( T_3 \) there is no “bib” node, and the “book” and “article” nodes are children of a “year” node that indicates their year of publication. We are interested in retrieving information by issuing the same query against all these data sources, even though the information is structured differently in each one of them. Therefore, we view all these XML trees as one tree \( T \) rooted at \( r \).

3.2. Query language

**Definition 3.1.** A Partial Tree-Pattern Query (PTPQ) is a triple \( Q = (\mathcal{P}, S, O) \), where:

(a) \( \mathcal{P} \) is a nonempty set of triples \((p, \mathcal{A}, \mathcal{R})\) called Partial Paths (PPs), \( p \) is the name of the PP. The names of the PPs in \( Q \) are distinct. Therefore, we identify PPs in \( Q \) with their names.

\( \mathcal{A} \) is a set of predicates of the form \( c = V \), where \( V \), the annotation of \( c \), is a set of values \( \{v_1, \ldots, v_k\} \), \( k \geq 1 \). The meaning of predicate \( c = V \) is that \( c = v_1 \) or \( \ldots \) or \( c = v_k \).

\( \mathcal{R} \) is a set of expressions of the form \( c_i \rightarrow c_j \) (child precedence relationship), \( c_i \Rightarrow c_j \) (descendant precedence relationship), and \( c_i =\Rightarrow c_j \) (descendant-or-self precedence relationship), where constructs \( c_i \) and \( c_j \) are distinct. In particular, \( \mathcal{R} \) comprises a descendant-or-self precedence relationship \( r =\Rightarrow c \), for every predicate \( c = V \) in \( \mathcal{A} \). The expression \( c[p] \) denotes the construct \( c \) in PP \( p \).

(b) \( S \) is a set of expressions of the form \( c_i[p_i] = c_j[p_j] \), where \( p_i \) and \( p_j \) are PPs in \( \mathcal{P} \). These expressions are called node sharing expressions. Roughly speaking, they state that the node labeled by construct \( c_i \) in PP \( p_i \) and the node labeled by construct \( c_j \) in PP \( p_j \) coincide (the two PPs share this node). Set \( S \) can be empty.

(c) \( O \) is a set of PPs in \( \mathcal{P} \). These PPs are called output PPs of \( Q \).
We graphically represent PTPQs using graph notation. Each PP of a PTPQ $Q$ is represented as a (not necessarily connected) graph of nodes identified with, and labeled by, the constructs of the PP. If a node $n$ in the graph has an annotation $V$ in $Q$, it is labeled by the predicate $c = V$ instead of the construct $c$. The name of each PP is shown by the corresponding PP graph. The names of the output PPs of $Q$ are shown boxed. Child, descendant and descendant-or-self precedence relationships in a PP are depicted using the arrows $\rightarrow$, $\Rightarrow$, and $\Rightarrow\Rightarrow$, respectively, between the corresponding nodes in the PP graph. In particular, descendant precedence relationships of the form $r \Rightarrow c$ and $r \Rightarrow\Rightarrow c$ in a PP are shown only with the presence of node $c$ in the PP graph. Variable names are preceded by a * sign, while values are shown between quotes. A node sharing expression $c[p_1] \equiv c[p_2]$ is represented by an edge between node $c_i$ of the PP graph $p_i$ and node $c_j$ of the PP graph $p_j$ labeled by the $\equiv$ symbol.

Suppose that we want to find the title and year of publications authored by “Mary” [19,20]. We are not interested to restrict the type of publication we are looking for, and actually we do not know what type of publications are recorded in the XML data. Further, assume that we know that title and author are not categorization features in our XML document(s), and therefore they should appear below any categorization element. We formulate this PTPQ as shown in Fig. 4. Symbols $x$ and $y$ denote variables, while $p_1$ and $p_2$ are the output PPs of the PTPQ. As another example, consider the query that finds additional authors of publications of which “Mary” is an author and also the title and year of those publications. In this case, assume that we expect author “Mary” and the other author to be sibling nodes and descendants of a publication node which has a descendant node “title”. We do not have any idea about the placement of node “year” besides the fact that it should relate in some way to the publication. This PTPQ is shown in Fig. 5.

The answer of a PTPQ is based on the concept of PTPQ embedding.

**Definition 3.2.** An embedding of a PTPQ $Q$ to an XML tree $T$ is a mapping $M$ of the constructs of the PPs of $Q$ to nodes in $T$ such that:

(a) An element $e$ of $Q$ is mapped by $M$ to a node in $T$ labeled by $e$; a variable $v$ of $Q$ is mapped by $M$ to a node in $T$ labeled by any element.
(b) The constructs of a PP in $Q$ are mapped by $M$ to nodes in $T$ that are on the same path.
(c) If a construct $c$ has an annotation $V$ in a PP $p$ (that is, a predicate $c = V$ is specified in $p$), then the image of $c[p]$ under $M$ has a child node labeled by a value in $V$.

![Fig. 4. PTPQ $Q_1$.](image)

![Fig. 5. PTPQ $Q_2$.](image)
We call image of a PP \( p \) in \( Q \) under \( M \), denoted \( M(p) \), the path from the root of \( T \) that comprises all the images of the constructs of \( p \) under \( M \) and ends in one of them. Notice that more than one PP of \( Q \) may have their image on the same root-to-leaf path of \( T \) (\( M \) does not have to be a bijection). The concept of image of a PP is extended to apply to PTPQs in a straightforward way.

We initially define the answers of a PTPQ on an XML tree as follows.

**Definition 3.3.** The answer set \( A \) of a PTPQ \( Q \) on an XML tree \( T \) is the set of subtrees of \( T \) formed by the images of the output PPs of \( Q \) under all possible embeddings of \( Q \) to \( T \). The subtrees comprise also the child value nodes of the elements. The subtrees in \( A \) are called answers of \( Q \) on \( T \).

Fig. 6 shows the images of PTPQ \( Q_1 \) of Fig. 4 under three of the possible embedding of \( Q_1 \) to the XML tree \( T \) of Fig. 3. The values of the elements are additionally included in the images for clarity. The images of the output PPs of \( Q \) in the figures are shown with thicker arrow edges. More specifically, Fig. 6a–c correspond to embeddings of \( Q_1 \) to the XML trees \( T_1 \), \( T_2 \) and \( T_3 \) respectively that constitute the tree \( T \) of Fig. 3.

Observe that the language is able to retrieve with one query the title and year of the publications of Mary from different parts of the XML tree, even though these parts structure the data in different ways.

The previous definition of the answer set of a PTPQ accepts any possible embedding of \( Q \) to \( T \). This generality allows embeddings that do not relate elements and values in the way the user was expecting when formulating the query. We call the answers corresponding to these embeddings meaningless answers. For instance, each of the images of \( Q_1 \) shown in Fig. 6 correctly corresponds to a publication (a book in this case) authored by “Mary”. However, this is not the case with the images of \( Q_1 \) in Fig. 7 under three other embeddings of \( Q_1 \) into \( T \). In each one of them, year and/or title values do not correspond to a publication authored by “Mary” even though these values appear in an answer with “Mary”. In Section 5, we will present a technique that excludes these subtrees and returns answers to the user that are meaningful.

Fig. 6. The images of \( Q_1 \) under three of the embeddings of \( Q_1 \) on \( T \). The answer of \( Q_1 \) in every image is shown with thicker arrows.

Fig. 7. The images of \( Q_1 \) under three of the embeddings of \( Q_1 \) on \( T \). The answer of \( Q_1 \) in every image is shown with thicker arrows.
4. Evaluating PTPQs using complete TPQs

We show now how PTPQs can be evaluated using TPQs. We first discuss index graphs for XML trees. Then, we use index graphs to construct a set of complete TPQs whose answers, taken together, form the answer of a given PTPQ. Besides allowing us to evaluate PTPQs, the complete TPQs of a PTPQ provide the basis for defining meaningful semantics for PTPQs in the next section.

4.1. Index graphs

Given a partitioning of the nodes of an XML tree $T$, an index graph for $T$ is a graph $G$ such that: (a) every node in $G$ is associated with a distinct equivalence class of element nodes in $T$, and (b) there is an edge in $G$ from the node associated with the equivalence class $a$ to the node associated with the equivalence class $b$, iff there is an edge in $T$ from a node in $a$ to a node in $b$. Index graphs have been referred to with different names in the literature including “path summaries”, “path indexes” and “structural summaries”. They differ in the equivalence relations they employ to partition the nodes of the XML tree which includes simulation and bisimulation [21,17] or even semantic equivalence relations [25]. Index graphs have been extensively studied in recent years in both the “exact” [11,21,6] and the “approximate” flavor [18,17]. A common characteristic of those approaches is that the index graph is used as a back end for evaluating a class of path expressions without accessing the XML tree. To this end, the equivalence classes of the XML tree nodes are attached to the corresponding index graph nodes.

For the needs of PTPQs we define index graphs where the equivalence classes are formed by all the nodes labeled by the same element in the XML tree. Fig. 8 shows the index graph $G$ of the XML tree $T$ of Fig. 3.

In contrast to other approaches, the equivalence classes of the XML tree nodes are not kept with the index graph. Therefore, PTPQs are ultimately evaluated on the XML tree. Even though the index graph for an XML tree is not a schema in the form of a DTD or an XML Schema, we take advantage of it in the same way schema information is exploited in relational databases. We use index graphs to support the evaluation of a PTPQ through the generation of a set of complete TPQs.

4.2. Complete TPQs for a PTPQ

If $G$ is the index graph of an XML tree $T$, we say that $T$ underlies $G$. Given a PTPQ $Q$ and an index graph $G$, $Q$ can be evaluated by computing a set of complete TPQs whose answers, taken together, are equal to the answer of $Q$ on any XML tree underlying $G$. By complete TPQ we mean a TPQ that involves only child relationships and no variables (and therefore, a TPQ that completely specifies a tree pattern). Intuitively, a complete TPQ satisfies both: the structural and value constraints of the PTPQ, and the structural constraints of the index graph.

**Definition 4.1.** Let $Q$ be a PTPQ and $G$ be an index graph. A complete TPQ (CTPQ) for $Q$ on $G$ is a TPQ $U$ without variables (wildcards) and descendant precedence relationships which is rooted at a node labeled by $r$ and satisfies the following conditions:

![Fig. 8. Index graph $G$.](image)
(a) There is a mapping $M$ from the nodes of $Q$ to the nodes of $U$ that respects paths, labeling elements, precedence relationships, and node sharing expressions. If $V_1, \ldots, V_k$ are the annotations of all the nodes in $Q$ that are mapped to the same node $n$ in $U$, $n$ is annotated by $V_1 \land \cdots \land V_k$. Two nodes in a path in $U$ are not labeled by the same element, and every leaf node of $U$ is the image of a node of $Q$ under $M$. The output nodes of $U$ are the images of the output PPs of $Q$. Notice that it is possible that all the nodes of two distinct PPs of $Q$ are mapped by $M$ to nodes on the same path in $U$.

(b) There is a mapping $M'$ from the nodes of $U$ to the nodes of $G$ that respects labeling elements and child precedence relationships.

Fig. 9 shows two of the CTPQs of the PTPQ $Q_1$ of Fig. 4 on the index graph $G$ of Fig. 8. The output nodes have their labels boxed. Clearly, a CTPQ can be seen as a PTPQ (without variables and descendant or descendant-or-self precedence relationships) where the node sharing expressions are defined by the common nodes of different root-to-leaf paths. The output PPs of the corresponding PTPQ are defined by the paths of the CTPQ that comprise output nodes. Then, we can define the answer of a CTPQ to be the answer of the corresponding PTPQ. We can now show the following proposition.

**Proposition 4.1.** Let $Q$ be a PTPQ, $G$ be an index graph, and $U_1, \ldots, U_k$, $k \geq 1$, be all the CTPQs of $Q$ on $G$. Let also $A, A_1, \ldots, A_k$ be the answer sets of $Q$, $U_1, \ldots, U_k$, respectively, on an XML tree underlying $G$. Then $A = \bigcup_{i \in [1,k]} A_i$.

Therefore, the answers of a PTPQ $Q$ on an XML tree $T$ can be computed by determining the set $\emptyset$ of all the CTPQs of $Q$ on the index graph of $T$ and by computing the answers of each CTPQ in $\emptyset$ on $T$.

Consider the XML tree $T$ (Fig. 3) and its index graph $G$ (Fig. 8). Consider also the PTPQ $Q_1$ (Fig. 4), and its CTPQs, $U_1$ and $U_3$, on $G$ (Fig. 9). One can see that the answer of $Q_1$ on $T$ shown in Fig. 6a is also an answer of CTPQ $U_1$. Similarly, the answer of $Q_1$ on $T$ shown in Fig. 7c is also an answer of CTPQ $U_3$.

Note that the approach presented in this paper can be easily extended to handle recursive XML trees (that is, XML trees that involve more than one node labeled by the same element in one path of the tree), and PTPQs that include multiple nodes labeled by the same element in the same PP. In this case, CTPQs for a PTPQ are generated using an index tree instead of an index graph. An index tree is a tree structure similar to an $1$-index [21] with the exception that no pointers to the data are stored in the index. The absence of cycles in the index tree allows one to deal with recursive XML trees and PTPQs.

5. Using complete TPQs to exclude meaningless answers

In this section, we assign semantics to our PTPQ language that returns meaningful answers. In contrast to previous approaches which exclude embeddings of the query to the data tree [24, 9, 19, 20], our approach excludes CTPQs of a PTPQ. In this sense, our approach relies both on data and on structural patterns of data, instead of relying exclusively on data.

---

2 1-indexes coincide with *strong DataGuides* when the data is a tree.
Based on the results of the previous section, we consider that, given an XML tree $T$ (and its index graph $G$), the answer of a PTPQ is the union of the answers of its CTPQs on $G$. However, some of these CTPQs may return meaningless answers. Consider, for instance, again, the PTPQ $Q_1$ (Fig. 4) and the XML tree of Fig. 3 along with its index graph $G$ in Fig. 8. The CTPQ $U_3$ (Fig. 9b) of $Q_1$ on $G$ returns (among others) the answer of Fig. 7c which is meaningless. Therefore, this CTPQ of $Q_1$ should not be used for computing the answers of $Q_1$. Analogously to query answers, we characterize a CTPQ of PTPQ $Q$ on $G$ as meaningful with respect to $T$ if it returns a meaningful answer on $T$. Otherwise, it is characterized as meaningless with respect to $T$. In order to formally define meaningful CTPQs we need to introduce a transformation for CTPQs.

### 5.1. A transformation for complete TPQs

Let $Q$ be a PTPQ, $T$ be an XML tree and $G$ be its index graph. Fig. 10 shows two CTPQs, $U$ and $U'$, of a PTPQ $Q$ on an index graph $G$. CTPQ $U$ comprises three subtrees $T_a$, $T_b$ and $T_c$. $T_a$ is the subtree of $U$ rooted at the node labeled by $a$, $T_c$ is a subtree of $U$ rooted at the node labeled by $c$, and $T_b$ is the subtree of $T_c$ rooted at a node labeled by $b$. Subtrees $T_a$ and $T_b$ can be empty (that is, they can trivially contain only their root node $a$ and $b$ respectively). The node labeled by $c$ can coincide with the root of $U$. However, the node labeled by $a$ cannot coincide with the node labeled by $c$, and the node labeled by $b$ cannot coincide with the node labeled by $c$ (that is, the node labeled by $c$ is an ancestor of the nodes labeled by $a$ and $b$). Labels $a$ and $b$ can be equal. Subtree $T'_a$ in $U'$ is a tree identical to $T_a$ except that its root is labeled by $a$ instead of $b$. CTPQ $U'$ can be obtained from $U$ by removing the subtree $T_c$ below the node labeled by $c$, and by making $T'_b$ a subtree of the node labeled by $a$. We define the transformation $TR$ on CTPQs as a transformation that transforms a CTPQ of the form of $U$ into a CTPQ of the form of $U'$. Notice that CTPQ $U'$ has at least one node less than CTPQ $U$.

We formally define meaningful CTPQs in the next subsection but we provide some intuition now on the transformation $TR$. Consider a CTPQ $U'$ resulting by applying $TR$ to a CTPQ $U$. Our intention is to characterize $U$ as meaningless with respect to $T$, and to exclude it from consideration in computing the answers of $Q$, if $U'$ returns an answer on $T$. To understand this idea, observe that there is a 1–1 mapping $f$ from the nodes of $U'$ to the nodes of $U$ that respects node labels and child precedence relationships (with the exception of the child precedence relationships from the node labeled by $a$ in $T'_b$). Then, the following proposition holds:

**Proposition 5.1.** Assume that CTPQ $U'$ results by applying transformation $TR$ to a CTPQ $U$. If $n'$ is the lowest common ancestor (LCA) of the nodes $n'_1, \ldots , n'_k$ in $U'$, and $n$ is the LCA of the nodes $f(n'_1), \ldots , f(n'_k)$ in $U$ then $n$ is not a descendant of $f(n')$ in $U$.

Since, there is an image of $Q$ under an embedding to $T$ (and therefore an answer of $Q$ on $T$) that closely relates the nodes as determined by $U'$, any image of $Q$ under an embedding to $T$ (and the corresponding answer) that relates the nodes in the looser way determined by $U$ is not meaningful, and should be excluded from generating an answer.

To clarify the use of transformation $TR$, we show next some applications of it on the CTPQs of our running example. We consider PTPQ $Q_1$ (Fig. 4) on index graph $G$ (Fig. 8) that underlies the XML tree $T$ (Fig. 3). Fig. 11 shows three CTPQs $U_1$, $U_2$ and $U_3$ of $Q_1$ on $G$. Dotted lines denote the subtrees $T_a$, $T_b$, and $T_c$ of
transformation $TR$ as they are graphically shown in Fig. 10. The CTPQ $U_2$ will be excluded from consideration in the evaluation of $Q_1$ on $T$ because TPQ $U_1$ returns an answer on $T$.

Similarly to $U_2$, CTPQ $U_3$ will be excluded from consideration. Notice that in the case of CTPQ $U_2$, the roots of $T_a$ and $T_b$ are labeled by the same element "book", while in the case of CTPQ $U_3$ they are labeled by different elements "book" and "article".

Fig. 12 shows applications of transformation $TR$ in sequence. The CTPQ $U_6$ is excluded from consideration because of the CTPQ $U_5$. Then, the CTPQ $U_5$ is also excluded because of the CTPQ $U_4$.

Finally, Fig. 13 shows some other applications of transformation $TR$ in sequence. Notice that $T_b$ in Fig. 13c (and consequently $T_b'$ in Fig. 13d) are empty. Observe that the CTPQ $U_8$ has an extra branch from the root with respect to CTPQ $U_7$.

5.2. Determining the meaningful complete TPQs

Next, we formally define the concept of meaningful CTPQ of a PTPQ on an index graph. Consider a PTPQ $Q$, an XML tree $T$, and its index graph $G$. Let $\mathcal{U}$ be the set of CTPQs of $Q$ on $G$. We define a binary relation $\prec$
on \( \mathcal{U} \) as follows: for every \( U, U' \in \mathcal{U}, U' \prec U \) if and only if \( U' \) can be obtained by applying a sequence of transformations \( TR \) to \( U \). Clearly, \( \prec \) is a strict partial order.

**Definition 5.1.** A CTPQ \( U \in \mathcal{U} \) is called meaningless with respect to \( T \) if there is another CTPQ \( U' \in \mathcal{U} \) such that (a) \( U' \prec U \), and (b) \( U' \) has an answer on \( T \). Otherwise, it is called meaningful with respect to \( T \).

We can now update the definition of the answer set of a PTPQ given in Section 3.2. We provide a new definition for the answer set of a PTPQ so that it comprises only answers of meaningful CTPQs. The new definition is based on Proposition 4.1 and Definition 5.1.

**Definition 5.2.** Let \( Q \) be a PTPQ, \( T \) be an XML tree and \( G \) be an index graph. Let also \( U_1, \ldots, U_k, k \geq 1 \), be the meaningful CTPQs of \( Q \) on \( G \) with respect to \( T \). If \( A, A_1, \ldots, A_k \) are the answer sets of \( Q, U_1, \ldots, U_k \), respectively, on \( T \), then \( A = \bigcup_{i \in [1,k]} A_i \).

Consider the CTPQ \( U_3 \) shown in Fig. 9b. As mentioned in Section 4.2, \( U_3 \), evaluated on the XML tree \( T \) of Fig. 3, returns the meaningless answer of Fig. 7c. CTPQ \( U_3 \) is also shown in Fig. 11 and it is characterized by Definition 5.1 as meaningless. Therefore, it will not be used to generate answers for the PTPQ \( Q_1 \) (Fig. 4) on \( T \). In contrast, CTPQ \( U_1 \) of Fig. 9a returns only the meaningful answer of Fig. 6a. CTPQ \( U_1 \), is also shown in Fig. 11. One can see that Transformation \( TR \) cannot be applied to \( U_1 \). Therefore, it is correctly characterized by Definition 5.1 as meaningful, and will be used to generate answers for the PTPQ \( Q_1 \) on \( T \). One can check that when it comes to evaluate \( Q_1 \) on \( T \), transformation \( TR \) excludes all CTPQs for \( Q_1 \) on \( G \) (Fig. 8) except the CTPQs \( U_1, U_4 \) and \( U_7 \) of Figs. 11–13, respectively, and the variations of those CTPQs where label “book” is replaced by “article”.

Since the meaningful CTPQs are TPQs, their evaluation can be implemented on top of an XQuery engine and benefit from the extensive optimization techniques that have been developed up to now for XQuery [15,4,7].

6. Analysis of previous approaches and comparison

In this section, we compare the semantics of our query language with the semantics of three other well known query languages for XML that aim at excluding meaningless answers [24,9,20,19,20]. In most practical cases, the information in the XML tree is incomplete (e.g. optional elements/values in the schema of the document are missing), or irregular (e.g. different structural patterns coexist in the same document). Therefore, we also take this parameter into account in our comparison.

Schmidt et al. [24] suggest the `meet` operator to let the users query an XML document without knowledge of the elements and the structure. Queries are sets of keywords to be matched against the values of the XML document. This approach exploits the structure of the XML tree and is based on merely computing the Lowest Common Ancestor (LCA) of the nodes in the XML tree that match the keywords. The computation of the LCAs is done bottom up. When the LCA of a set of nodes that match the keywords is computed, these nodes are excluded from further consideration. The meet operator might fail to return a meaningful answer when a node is a descendant of another node of similar type (logical hierarchy) and the information in the XML tree is not complete. Consider, for instance, the XML tree of Fig. 14 and a keyword query consisting of the keywords “Mary”, “title” and “year”. This approach considers only keywords that are values but we allow also keywords that are elements in this example as this does not affect the computation of LCAs. The `meet` operator will fail to return the subtree rooted at the node labeled by `bib` which is the meaningful answer. The reason is that another LCA node is identified first (the node labeled by “book”) and the subtree rooted at this node is excluded from further consideration. The meet operation will also fail to exclude meaningless answers in case of incomplete information even for a flat XML tree (that is, a tree that does not contain logical hierarchies). Consider, for instance, the same keyword query and the XML tree \( T_1 \) of Fig. 3. The meet operator will return the meaningless answer shown in Fig. 7b.

XSEarch [9] is a semantic search engine for XML. It uses a simple query language that allows keyword specifications (values and/or elements) and a primitive structural restriction (a node labeled by an element keyword has a descendant node labeled by a value keyword). The answers are subtrees that contain the keyword...
labeled nodes. XSEarch uses the concept of Interconnection Relationship to capture the meaningful XML subtree for a set of nodes that match the keywords. Two nodes $n_1$ and $n_2$ are interconnected if the path from $n_1$ to $n_2$ through their LCA does not contain two distinct nodes labeled by the same element. Nodes $n_1$ and $n_2$ can have the same label though. The interconnection relationship is extended to multiple nodes through an all-pair or a star $n$-ary relationship. XSEarch allows only query answers where the nodes matched by the keywords are all-pair or star related. Intuitively, nodes in the XML tree represent entities and element labels represent their type. Nodes with the same label represent entities of the same type. Descendant nodes of a node $n$ are assumed to belong to the entity $n$ represents. Two nodes that are meaningfully related should not belong to different entities of the same type. XSEarch rarely fails to return a meaningful answer. However, it usually fails to exclude meaningless answers even if the XML tree is flat, and does not have incomplete information. Consider the keyword query $Q_1$ specifying the elements “title” and “year” and the value “Mary” on the XML tree of Fig. 3. XSEarch fails to exclude the meaningless answer of Fig. 7c. The reason is that any two of the nodes that match the keywords are interconnected (in contrast, XSEarch succeeds in excluding the meaningless answer of 7a because this one contains two nodes which are both labeled by “bib”). Similarly, query $Q_1$ issued against the XML tree of Fig. 14 which has a logical hierarchy fails to exclude the several meaningless subtrees-answers rooted at the node labeled by the “book”.

Li et al. [19,20] extend XQuery to enable users to query XML documents without full knowledge of the structure. This work is closer to ours compared to the previous two because it allows the user to specify extensive structural restrictions in a query besides keywords. To compute a query, this approach finds the LCA node of the set of nodes that match the keywords, and treats the subtree rooted at this node as the context for query evaluation. It employs a particular version of LCA, called Meaningful Lowest Common Ancestor Structure (MLCAS). The MLCAS of two nodes $n_1$ and $n_2$ (and therefore that of any superset of those two nodes) does not exist if two other nodes of the same type (that is, nodes labeled by the same element) have an LCA which is a descendant of the LCA of $n_1$ and $n_2$. The MLCAS approach fails to return meaningful answers when the XML data contains logical hierarchies even if there is no incomplete data in it. Consider, for instance, query $Q_1$ specifying the elements “title” and “year” and the value “Mary” on the XML tree of Fig. 15. Under the MLCAS semantics, the answer set of the query does not contain the subtree-answer rooted at the node labeled by element “bib” (shown with bold arrows in Fig. 15), which, intuitively, is the answer the “most related” to query $Q_1$. The answer set will contain only the subtree-answer rooted at the node labeled by “reference”. When the XML data is incomplete, the MLCAS approach fails to exclude meaningless answers. For instance, the keyword query $Q_1$ specifying the elements “title” and “year” and the value “Mary” on the XML tree of Fig. 3 will return the meaningless answer of Fig. 7b because there are articles in the XML tree $T$ that have only a title (and no author) and articles that have an author (and no title). Another drawback of this approach is that the semantics for the keyword queries (MLCAS) is different than the semantics for the structural queries (XQuery). Therefore, the structural restrictions cannot be taken into account in determining the meaningful answers of a query in the first place. If a meaningful answer of a query is not contained in the subtrees returned by the keyword search part of the query, it cannot by recovered by further evaluating
the structural part of the query. Notice that, in contrast, in our approach both the structural restrictions and
the keywords in a query determine the meaningful TPQs that, in turn, compute the answers of a query.

Our approach successfully returns all the meaningful and eliminates all the meaningless answers of the
examples discussed in this section. Its success is due to the original way it uses to evaluate the meaningful
answers of a query. Previous approaches identify meaningful answers by operating locally on the data by com-
puting LCAs of nodes in the XML tree. In contrast, our approach operates globally on structural summaries
of data (index graphs) to compute meaningful TPQs. This overview of data gives an advantage to our approach
compared to previous ones.

7. Experimental evaluation

We implemented our approach (abbreviated as PTPQ), and we experimentally compared it to previous
approaches on two aspects: the quality of the returned results, and the efficiency of their computation.

7.1. Quality

In order to assess the quality of our approach, we implemented the three approaches discussed in Section 6
(Meet [24], XSEarch [9], and MLCAS [19,20]). We ran detailed experiments to compare their Recall (defined
as the proportion of relevant materials retrieved) and Precision (defined as the proportion of retrieved materials
that are relevant). These parameters have been used for years for measuring the quality of keyword search in
information retrieval systems. As our language is a database language, we did not use ranking functions com-
bined with threshold values to trade recall for precision (or vice versa).

7.1.1. Experimental setting

We used real-world DBLP data of the size of 324 MB collected from http://dblp.uni-trier.de/xml/ in May
2006. To simplify the document for the experiments, we retained only three publication types: “book”, “arti-
cle”, and “inproceedings”. For each publication type, we retained only the properties “title”, “authors”, and
“year”. As the original DBLP data is flat, for evaluation purposes, we restructured it into three types of data
sets that comply respectively with the schemas shown compactly as dags in Fig. 16. All subelements of pub-
lications in the three schemas are optional as in the original DBLP schema. Publications of the schema Type 1
do not have references. Publications of schemas Type 2 or Type 3 may have references. We consider also ref-
ences to be publications. Therefore, schemas Type 2 and 3 contain logical hierarchies. One difference
between schema Type 2 and Type 3 is that publications of schema Type 3 are categorized by year.

Besides the structure of the document, the “incompleteness” of the data also affects the effectiveness of the
keyword based searches. We define a publication in the data set as complete if it has all the subelements “title”,
“year”, and “author”, otherwise it is incomplete. For the experiments, we considered data sets that have dif-
ferent percentages of incomplete publications.
The data sets for the three schema types are generated as follows. A program loads a set of sampled "book", "article", and "inproceedings" elements each with three subelements "author", "title", and "year" from the original DBLP data. Another program randomly chooses a set of publications among them for removal of some of their subelements. One or at most two subelements can be removed from each publication. The percentage of incomplete publications for different publication types can be specified through input parameters. Finally, an XML creator reassembles the publications to an XML document. The structure of the generated XML file, determined also by an input parameter, can be any one of the three types shown in Fig. 16.

As the distinction between keywords that are values and keywords that are elements is insignificant for the semantics of the queries in all approaches, in our experiments, we query only for elements. We used keyword queries that comprise at least two of the elements "title", "year", and "author". We also ran experiments on queries with more than three keywords and the results were similar. We also considered two of the previous keyword queries with structural restrictions. We used these queries to experimentally compare only our approach (PTPQ) and the MLCAS approach since these are the only two that allow the specification of non-primitive structural restrictions.

For all three LCA-based approaches, we consider that the answer of a query is the subtree whose leaves are the nodes that match the keywords and whose root is their LCA (the way it is defined in each approach). Thus two distinct matchings of the keywords with the same LCA determine two different answers. For each query and each type of data set, we wrote a fully specified query in XQuery that expresses what the user is seeking. We used the answers of these queries as a reference for computing precision and recall.

For each query and each type of data set, we have run the four approaches on six XML documents with increasing percentage of incomplete publications in the range from 0 (all the publications are complete) to 50% (half the publications are incomplete).

We ran the experiments on a Pentium 2.40 GHz computer with 512 MB of RAM running Windows XP Professional. We implemented all keyword search techniques in Java and used the SAX API of the Xerces Java Parser for the parsing of XML files. Berkeley DB XML 2.2.13 was used to store XML files and run XQuery.

7.1.2. Experimental results for keyword queries without structural restrictions

We first consider keyword queries without structural restrictions. Fig. 17 shows precision and recall of the two keyword query \{author, year\} for the three types of documents varying the percentage of incomplete publications in the documents.

Both XSEarch and PTPQ have perfect recall on all types of documents both for complete and incomplete data. Meet has also perfect recall on Type 1 and 2 documents but performs slightly worse on Type 3 documents when the data is not complete. MLCAS has also perfect recall on Type 1 documents. In contrast, its recall is degraded on Type 2 documents and it drops below 60% on Type 3 documents both for complete and incomplete data. This is due to the fact MLCAS cannot handle the logical hierarchies appearing in Type 2 and 3 documents.

PTPQ shows perfect precision on Type 1 document (no logical hierarchies). Its precision starts above 60% for complete data and goes slightly up as the percentage of incomplete publications increases on Type 2 and 3 documents.

Fig. 16. Three schemas for the DBLP data: (a) Type 1, (b) Type 2, (c) Type 3.
documents. The opposite trend is followed by the precision of Meet and MLCAS on all types of documents. They start at 100% with complete data and drop as the percentage of incomplete publications increases. The precision of XSEarch is, in general, low and is not affected significantly by the increase of the percentage of incomplete information.

Fig. 18 shows the precision and recall of the two-keyword query {title, author} for the three types of documents varying the percentage of incomplete publications in the documents. We omit the plots of the query {title, year} as they are analogous to those of the query {title, author}.

All four approaches shown in Fig. 18 are similar trends to those shown in Fig. 17. Meet and XSEarch show on the average even lower precision. Their recall is perfect for all types of documents both for complete and incomplete data. Interestingly, the recall of MLCAS improves when the percentage of incomplete publications increases, reaching 100% for a percentage of 50% of incomplete publications. The reason is that when the number of incomplete publications increases, a number of “book”, “article”, and “inproceedings” elements (which are missed anyway by the MLCAS approach) are not anymore correct answers.
Fig. 19 shows the precision and recall of the three-keyword query \{title, author, year\} for the three types of documents varying the percentage of incomplete publications in the documents. The trends are similar to those of two-keyword queries with a slight degradation of the recall of Meet, and an average degradation of the precision of XSEarch and Meet.

7.1.3. Experimental results for keyword queries with structural restrictions

We now consider queries that involve also structural restrictions. We use two of the previous keyword queries where the “author” and “title” keyword elements are both child nodes of some (the same) element. This structural restriction can be formulated on the keyword queries \{author, title\}, and \{title, author, year\}. We call the first one $Q_{s2}$ and the second one $Q_{s3}$.

Fig. 20 shows the precision and recall of query $Q_{s2}$ for Type 3 documents varying the percentage of incomplete publications in the document (for Type 1 document the precision and recall are perfect for both
approaches). Both approaches show for $Q_{s2}$ the same recall as for the corresponding query without structural restrictions (Fig. 18), which for the PTPQ approach is 100%. Both approaches improve their precision achieving a perfect one.

Fig. 21 shows the precision and recall of query $Q_{s3}$ for Types 2 and 3 documents varying the percentage of incomplete publications in the document. The PTPQ approach has perfect recall. The MLCAS approach has for $Q_{s3}$ the same recall as for the corresponding query without structural restrictions (Fig. 19). Both approaches improve their precision but the improvement is more important for PTPQ.

In summary, MLCA shows good precision which can be improved with structural restrictions. However, its recall is low (it falls below 60% in some cases). Its recall cannot be improved when additional structural restrictions are imposed since the semantics for the keyword part of the query is different than that of the structural part of the query. Therefore, answers missed in the evaluation of the keyword search part of the query
cannot be recovered during the evaluation of the structural part. PTPQ does not show this drawback. Its recall is perfect with and without structural restrictions, while additional structural restrictions improve its recall.

7.2. Performance

In order to assess the performance of our approach, we compared it to the MLCAS approach for two reasons: (a) MLCAS also allows the specification of structural constraints, and (b) MLCAS has been embedded...
into Timber [15], an XML database management system. In order to guarantee a unique experimental comparison environment, we used Timber also for the evaluation of the meaningful CTPQs of our approach. In this section, we present the experiments conducted to evaluate the performances of PTPQ approach and MLCAS approach on generating meaningful answers and report results obtained.

7.2.1. Experimental setting

We compare the time cost of evaluating an MLCAS-embedded XQuery, with that of the PTPQ approach, which generates a set of meaningful CTPQs, and then evaluates their corresponding XQuery expression.

We used original DBLP datasets. We retained only the properties of “year”, “author”, “title”, “publisher”, and “ISBN” for each publication as these are the ones that we used in queries, and removed other properties such as “volume” and “pages”. In the experiments we used five different sizes of datasets: 95 KB, 21 MB, 29 MB, 95 MB and 148 MB. We used the same platform as the one used for the quality experiments.

Running queries with several keywords and no structural constraints in the MLCAS approach takes a lot a time when the data set is larger than 90 KB. We used the following four queries in the experiments:

Query 1: A pure two keyword query (“www” and “title”).
Query 2: A three keyword query (“title”, “year” and “ISBN”) with a variable and a structural constraint: “ISBN” is a child of the variable. Note that the structural constraint in the MLCAS approach is specified within the XQuery part of the query.

Fig. 22. Performance comparison for Queries 1–4.
Query 3: A three keyword query ("title", "article" and "year") with a structural constraint: "title" is a descendant of "article". The structural constraint in the MLCAS approach is specified within the MLCAS function part of the query.

Query 4: A two keyword query ("title" and "inproceedings") with a structural constraint: "title" is a descendant of "inproceedings".

For each approach, we ran the four queries on the five data sets. The running time of PTPQ queries consists of the time for generating meaningful CTPQs and the time for evaluating these CTPQs (translated into XQuery expressions). The running time of an XQuery expression on Timber reflects the execution of its physical plan and does not include the time for parsing the expression. We run each query five times on each data set, and we recorded the average running time. The running time of MLCAS queries is merely the time Timber records for these queries.

7.2.2. Experimental results

Fig. 22 respectively report the execution time of the MLCAS and PTPQ approaches. Note that the y-axis is on a logarithmic scale. We can see that the time for generating meaningful CTPQs is insignificant, only around 1% of the total evaluation time for the PTPQ approach. We can also see that the execution time of MLCAS approach is larger than that of PTPQ approach in some cases by over two orders of magnitude. Such a big difference in execution time is expected as the MLCAS approach works solely on data while the PTPQ approach exploits index graphs whose size is much smaller than that of underlying data. Moreover, we can see that the PTPQ approach scales much better than the MLCAS approach.

Our experiment results show that the PTPQ approach outperforms the MLCAS approach in both query performance and quality of results.

8. Conclusion

Current applications export and exchange XML data. In this context, a major challenge is the querying of the data when the structure is not fully known, and the integrated querying of multiple data sources that export data with structural differences and irregularities.

We have introduced a query language with wildcards that allows partial specification of a tree pattern. Our query language flexibly allows the specification of the full spectrum of queries from structureless keyword queries to complete tree-pattern queries. We have suggested an original approach for assigning semantics to this language. In contrast to previous approaches that operate locally on data, our approach operates globally on structural summaries of data to extract tree patterns. An experimental evaluation of the quality of our approach shows that it has a perfect recall for both XML documents with complete and incomplete data. It also shows better precision, in most cases, compared to approaches with similar recall. Our approach can be directly implemented on top of an XQuery engine. An experimental evaluation of its efficiency shows that it outperforms and scales better than the only previous approach that allows for structural constraints in the queries and has been integrated into an XQuery engine.

An extension direction of the current work involves using multiple query optimization techniques to identify commonalities between the meaningful complete TPQs of a PTPQ. These commonalities can then be exploited in deriving an efficient "global" plan that evaluates all the meaningful complete TPQ together.

References

Dimitri Theodoratos is an Associate Professor in the Department of Computer Science of the New Jersey Institute of Technology (NJIT). He holds a Ph.D. from the University of Paris at Orsay, and a Masteres degree from the Ecole Nationale SupTrieure de TTlTcommunications de Paris, in Computer Science, and a Diploma in Electrical and Computer Engineering from the National Technical University of Athens. Prior to NJIT, he has taught at the National Technical University of Athens, the University of Ioannina, Greece, and the University of Paris at Orsay, and has been a postdoctoral fellow at INRIA in Paris and at RAL in Oxfordshire, UK. His research interests include data base theory, data integration, data warehousing, XML, query processing and optimization, and semantic web. He has published many papers in these areas and has served on the program committees of relevant conferences.
Xiaoying Wu is currently a Ph.D. candidate in the Computer Science Department of New Jersey Institute of Technology under the supervision of Professor Dimitri Theodoratos. She received her B.S. degree in Computer Science from Central South University in China and an M.S. degree in Computer Science from the National University of Singapore. She has published and conducted research on issues related to semistructured data and XML, keyword-query search, query semantics, query processing and optimization.