Journal of Computing and Information Technology - CIT 19, 2011, 2, 83–91
doi:10.2498/cit.1001006

GXQuery: Extending XQuery for Querying Graph-structured XML Data

Hongzhi Wang and Jianzhong Li

Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, China

XML data can be naturally modeled as a graph. Existing query languages to XML can only express queries of matching XML document with a tree-structured schema with structural and value constraints without the consideration of graph features. The ability of such query languages cannot satisfy various requirements of querying graph-structured XML data. In this paper, GXQuery is presented as an extension of XQuery, an XML query language recommended by W3C, to express more flexible query on graph-structured XML. GXQuery expressions can match XML document with graph-structured schema with not only structural and value constraints, but also topological constraints.

Keywords: XML, XQuery, query language, topological

1. Introduction

In some applications, XML data can be naturally modeled as graph structure. For an example, Figure 1 shows a graph structure of an XML document in Figure 2. This is a document about the relationship between authors and publications. Such document adapts to graph structure since one paper may have more than one author and one author may have more than one paper.

```
<bib>
<conference id="c1">
<name>n1</name>
<paper author="person3">
<title>t1</title>
</paper>
<paper author="person2">
<title>t2</title>
</paper>
</conference>
<conference id="c2">
<name>n2</name>
<paper author="person2">
<title>t3</title>
</paper>
<paper author="person3">
<title>t4</title>
</paper>
</conference>
<journal id="j1">
<name>n3</name>
<paper author="person2">
<title>t2</title>
</paper>
<paper author="person1">
<title>t4</title>
</paper>
</journal>
<persons>
<person id="person1" editor="j1">
<name>p1</name>
<address>a1</address>
</person>
<person id="person2" PCnumber="c2">
<name>p2</name>
<address>a2</address>
</person>
<person id="person3" editor="c1">
<name>p3</name>
<address>a3</address>
</person>
</persons>
</bib>
```

Figure 2. bib.xml.

Query processing on graph-structured XML data brings new challenges. One of them is that current query language can not satisfy all requirements of query description of graph-structured XML documents.

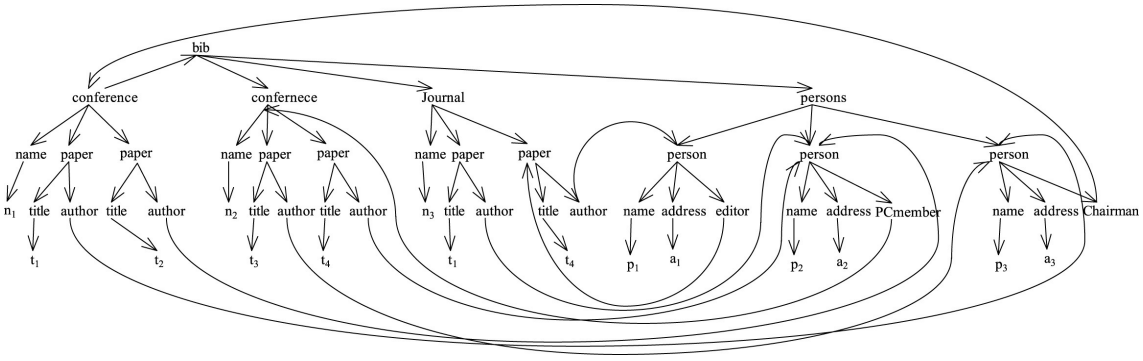


Figure 1. An example of graph structure.

Current query languages [2] for XML do not support the following features for graph-structured XML document.

- Current query languages for XML are designed for tree-structured XML data and do not support the matching of schema in form of general graph. Even though XPath can express a node with multiple parents by multiple constraints with axis "parent", it cannot express a graph with cycles.
- Current query languages for XML do not support topological constraints. Topological constraint in the query to a graph means that the required graph must have some topological constraint with a given graph. For example, in Figure 5(d), subgraph G_A in Figure 5(b) and subgraph G_B in Figure 5(b) are overlapping. Topological relationships between two graphs include connect, overlapping, contain and disjoint.

In real applications, these two features for querying graph-structured XML data have their applications. Two applications are shown in following examples.

- Retrieve the person who has published paper in the conference with himself as a PCmember. Performing such query in the graph in Figure 1 is to retrieve person element in a circle "$person \rightarrow Pcmember \rightarrow conference \rightarrow paper \rightarrow author \rightarrow person$". Note that the last person and first one should be the same element.
- Retrieve the conference which shares at least one paper with same title and authors as a journal. Such query uses a topological relationship "overlapping" as a constraint. Performing such query in the graph in Figure 1 is to find subgraphs containing conference and sharing common parts of author information with some subgraph containing information about journal.

In this paper, in order to describe queries on graph-structured XML data effectively, we extend XQuery [3], a query language for XML recommended by W3C by adding these features of graph representation and topological constraints to XQuery. We define a new data type XGraph as an extension of XPath to express structural constraints as a general graph, which may contain some circles. We also define topological constraints to make XQuery expressions support topological queries. These two new features can be embedded in current XQuery expression seamlessly.

The contributions of this paper includes:

- XPath is extended to be XGraph to support the description of general directed graph. XGraph can express a kind of graphs in a flexible and concise way. The extension reserves original features of XPath. Additionally, XGraph can be embedded in original XPath expression as structural constraint.
- With the support of XGraph, topological constraint is added to XPath and XQuery. With such extension, queries such as "retrieve graphs overlapping with graph B" can be expressed. As far as we know, this is the first paper that considers this problem.
- A labelling-scheme-based processing strategy of GXQuery is presented in this paper.

This paper is organized as follows. In Section 2, some background knowledge is presented. In Section 3, we give the description of XGraph and topological constraints. Some use cases about extended XQuery are presented in Section 4. We design preliminary implementation of GXQuery in Section 5. In Section 6, we give an overview of work related to this paper. We draw the conclusions in Section 7.

2. Preliminaries

In this section, we briefly introduce graph-structured XML model and some terms used in this paper.

2.1. Data model

XML data is often modeled as a labelled tree: elements and attributes are mapped into nodes of the tree; directed nesting relationships are mapped into edges in the tree. A feature of XML is that there may be an IDREF between two elements representing their reference relationship [11]. With this feature, XML data can be modeled as a labelled digraph: elements and attributes are mapped into nodes of the graph; directed nesting and reference relationships are

mapped into edges in the graph. An XML fragment is shown in Figure 2, which can be modeled as the graph shown in Figure 1. Note that the graph in Figure 1 is not a DAG.

2.2. XPath and XQuery

XPath [5]is a path description language presented by W3C. The unit of XPath is called a *step*. A step generates a sequence of items and then filters the sequence by zero or more predicates. The result of the step consists of items that satisfy the predicates. Such a step has two parts: an axis, which defines the "direction of movement" for the step, and a node test, which specifies the node kind and/or name of the nodes to be selected by the step. For example, to retrieve the title of the paper published in a conference "n1", the XPath expression is $bib/conference[name = "n1"]/paper/title$. The results of performing this query on XML document in Figure 2 is $t1$ and $t2$.

XQuery [3] is a query language recommended by W3C. It uses XPath to express complex path and supports flexible query semantics. XQuery has "For-Let-Where-Return"(FLWR for brief) structure. For example, query

$$
\begin{aligned}
&for\ \$i{=}document(\text{“}bib.xml\text{”})//\\
&\qquad person[PCmember]\\
&let\ \$j = \$i/addres\\
&where\ \$j = \text{“}a2\text{”}\\
&return < name > \$i/name < /name >
\end{aligned}
$$

is to retrieve the name of a person with address "a2" who has been a PCmember of some conference. The result of this query is an XML fragment "$< name > p2 < /name >$".

2.3. Labelling schemes

The labelling scheme for a graph-structured XML document is to judge the structural relationship between any two nodes in a graph.

In XML document without accessing other information. Such that subgraph queries can be processed efficiently. The labelling scheme used in this paper is an extension of that in [9].

The reachability labelling scheme can be generated in the following steps:

- Each strongly connected component in G is contracted to one node to convert G to a DAG D.
- An optimum tree covering T of the DAG D is found. A depth-first traversal from the root of T accesses all nodes to generate the post-order of each node. Note that during the traversal, when a node n_C generated from a strongly connected component $C \subset G$ is accessed, if the post order of last accessed node is pc, then $pc+1, pc+2, \ldots, pc+|V_c|$ are assigned to n_C (where V_C is the number of nodes in C). Then, each node $n \in T$ is assigned a number id and an interval $[x, y]$, where id and y are both the post order of n; x is the smallest post order of descendants of n in T.
- All the nodes in D are traversed in the reversed topological order. When a node n is met, the interval sets of n's children in D are copied to that of n. Then intersected intervals in the interval set of n are merged.
- For each node in C, its interval set is that of n_C; its id is one of the ids of n_C. Note that each node in C has a different id.

When such steps are finished, each node n in G is assigned a number $n.id$ and a set of intervals I_n. A node a reaches a node b (no matter whether a and b are in the same strongly connected component) if and only if $b.id$ belongs to some interval of I_a. With slight modification of attaching the interval $[i, i]$ of each node with id i to its parent, the adjacent labelling scheme is generated and the judgement condition is the same as that of reachability labelling schemes.

3. Description of GXQuery

In this section, we describe how to extend XQuery to support querying graph flexibly. A basic idea of the enhancement is to describe the query to a graph flexibly. By extending XPath, we present XGraph. Similar to XPath expression describing matching rules of a path, XGraph describes matching rules of a graph.

In this section, at first, we will present the description of XGraph. Then, we describe topological relationships between XGraph objects. At last, we present how to embed XGraph features into an XQuery expression.

3.1. The definition of XGraph

In this subsection, we will present the definition of XGraph. XGraph is an expression language to describe a special class of graph matching some patterns. An example of graph structure of an XML document is shown in Figure 3.

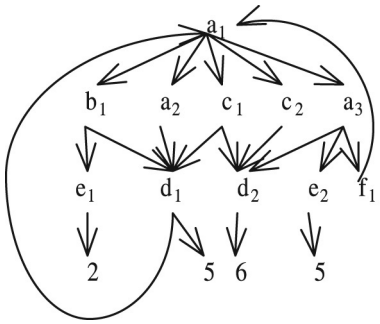


Figure 3. Example graph.

In XGraph, we use XPath as the backbone to describe general graph. Each XGraph expression can be represented as a graph, as is called *query graph*. There are two kinds of nodes in query graph, nodes as result and nodes as structural constraint. Similar to XPath, we use "[]" to represent "existing" a branch or subgraph in query expression. We use "<>" to represent a schema to be matched and exist in the result of this XGraph expression. If a tag is not in any bracket, it represents structural constraint. In a simple example query with query graph in Figure 4(a), since a at the top of the figure has more than one child/descendant in the query, all its children/descendants are in brackets. It is different from XPath. An XPath expression can only represent the query requiring a special kind of nodes, while an XGraph expression can represent the query requiring a special kind of subgraphs. For example, an XPath expression $a/b/c$ will retrieve nodes with tag c and incoming path a/b. An XGraph expression $< a/b/c >$ will retrieve all subgraphs with schema $a/b/c$. XGraph has the same semantics as XPath when nodes in XML document matching only one node in query should be returned as results. For example, XGraph expression $a/b/c$ has the same meaning as XPath expression $a/b/c$.

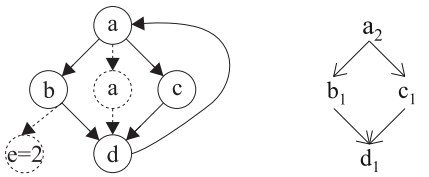
In order to describe circle or node with multiple parents/ancestors, we add variables binding to XGraph expression. "%" is used as the suffix of variable in XGraph expression. The variable in brackets "()" following a node n is the variable binding to n.

We use an example to illustrate XGraph. The structure of a graph query is shown in Figure 4(a). Solid lines and circles in the query graph represent the nodes and relationships to be matched as result in the graph. Dotted lines and circles represent that the nodes and relationships are constraints. Single line represents parent-child relationship. Double lines represents ancestor-descendant relationship. The expression of the query graph in Figure 4(a) is as follows.

Query 1 $a(\%a)[a/\%d] < //c//\%d > / < b[e = 2]/d(\%d)\%a >$

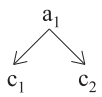
where $\%a$ and $\%d$ are variables binding to query nodes with tag a and d, respectively. '/' and '//' in Query 1 represent parent-child relationship and ancestor-descendant relationship, respectively. The result of processing Query 1 in Figure 3 is shown in Figure 4(b).

Another case is that in the graph, it is required

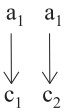


(a) Query 1

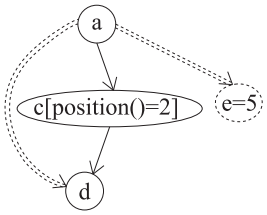
(b) Result 1



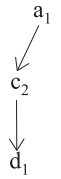
(c) Result 2



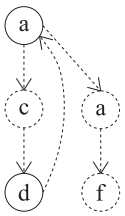
(d) Result 3



(e) Query 4



(f) Result 4



(g) Query 5

Figure 4. Example queries.

that multiple nodes match one node in the query while other nodes in the query only match just one node in the document graph. We use "*" to represent multiple-matching part of the query. For example, a query

Query 2 $< a/ < c > * >$

will return result shown in Figure 4(c).

As a comparison, a query

Query 3 $< a/c >$

will return results as is shown in Figure 4(d).

The result of XGraph expression is a graph. Note that in the result, both ancestor-descendent and parent-child relationships between two matching nodes are both represented as parent-child relationship.

XGraph reserves all the features of XPath, such as axis, structural, value and position constraints. For example, query

Query 4 $< a(\%a)/c[position() = 2]/$
$d[::ancestor\%a[//e = 5]] >$

represents the query within form of graph in Figure 4(e). The result of Query 4 is shown in Figure 4(f).

XGraph expression can be embedded in an XPath expression as structural constraint. For an example, query

Query 5 $a/c/d[a(\%a)/a/f\ /\%a]$

represents the query graph in Figure 4(g). The result is d_1.

3.2. Topological constraints of XGraph

In this subsection, we present the constraints among XGraph objects. Besides traditional constraints of value, structure and position, we present a novel constraint, topological constraint of XGraph object. There are five kinds of topological relationship between two XGraph object. Their semantics, expressions and examples are shown in Table 1. The topological relationships are illustrated by an example. Three subgraphs G_A, G_B and G_C are shown in Figure 5(a), Figure 5(b) and Figure 5(c), respec-

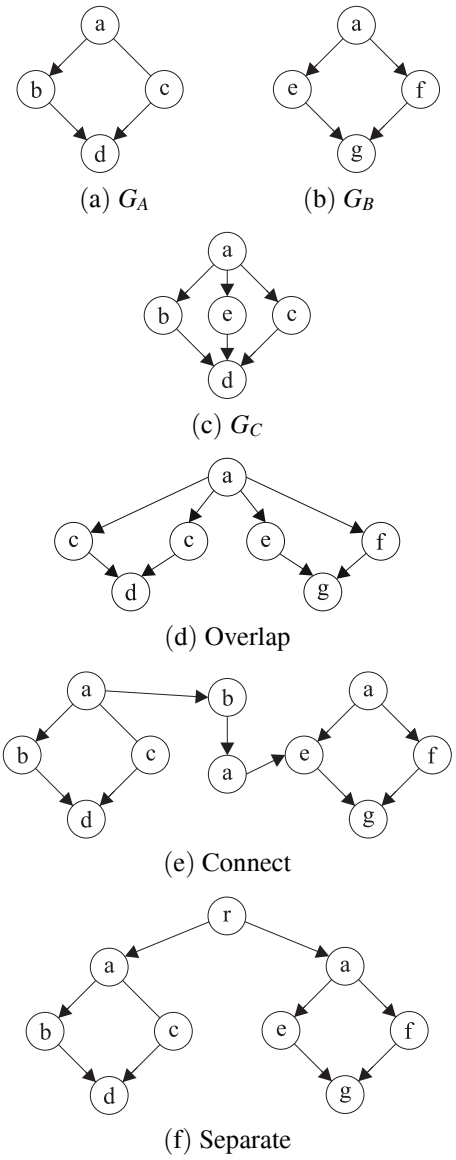


Figure 5. Example of constraints.

relationship	semantics	expression	example
connect	two graphs connected with some edge	G_A CONNECTED G_B WITH a/b	Figure 5(e)
overlap	two graphs have some common part	G_A CONNECTED G_B WITH $< a >$	Figure 5(d)
contain	one graph contains another graph	G_C CONTAIN G_B	Figure 5(c)
be contained	one graph is contained by another graph	G_B IN G_C	Figure 5(c)
disjoint	there is no path between any nodes of these two graphs	G_A DISJOINT G_B	Figure 5(f)

Table 1. Topological relationships.

tively.

The "example" column in Table 1 means if subgraphs are in such relationship, then expression in "expression" column is true.

Topological constraint can be embedded in XGraph expression as a filter. The usage of such expression is the same as position and value constraints in XPath expression. For example, query

Query 6 $< a/c/d > [graph()SEPARATE < //a/e >]$

represents subgraphs with schema $a/c/d$ which have no common node with any subgraph with schema $//a/e$ in the same graph.The result of performing this query on the graph in Figure 3 is only node d_2. d_1 is not the result because that it is connected with a subgraph $a1/e1$ with a common node $a1$.

XPath can express connection relationship between two nodes but not graphs. This is different from the connection relationship in XGraph. In XGraph, the constraint "A CONNECT B WITH a/b" is true if there is a path "a/b" between any node in *A* and any node in *B*.

3.3. Embedding XGraph and topological constraints into XQuery

XGraph expressions and topological constraints can be embedded XQuery seamlessly. In this subsection, we will show how to embed XGraph expressions and topological constraints into XQuery expression. We define XQuery extended by these two features of XGraph expression and topological constraint as *GXQuery*.

An XGraph object can be used in the similar way as an element described by XPath. A variable with suffix "#" is used to represent an XGraph object. Note that XGraph is different from XPath because a variable binding to an XGraph object represents a subgraph while in an XPath expression, a binding variable represents just one node. In order to distinguish the difference of such semantics, we add a keyword "AS" to define the schema to matching. The keyword "IN" is used to describe whose subgraph this XGraph object is. The expression after *AS* should be an XGraph expression. The expression after *in* should be an XGraph object or variable. A document is treated as an XGraph object.

When there is only one matching node in an XGraph expression, a variable with suffix "$" can also bind to such expression. The semantics of such binding is the same as that of XPath.

For example, we suppose the XML document corresponding to the graph in Figure 3 is "text.xml", query

Query 7 *FOR#G AS* $< a < //b > //c/d >$ *IN document*("*test.xml*")
FOR #SG AS $< a//c/d >$ *IN #G*
FOR $d *AS*$ $a//c/d$ *IN #SG*
RETURN $<$ *result* $>$ $d $ $<$ */result* $>$

returns result

```
<result><d>5</d></result>
<result><d>6</d></result>
<result><d>6</d></result>
```

Since an XGraph variable represents a graph instead of a node, if it exists in a "return" clause, there is an XGraph variable, the graph should be represented in the form of an XML fragment with "ID" and "IDREF". The number of id is the same as that in original XML document. Note that it is different from variables binding to XPath expression, the variable binding to an XGraph expression in "return" clause returns only the elements in XML documents matching nodes in "<>" in the XGraph expression with value as the result. If a complex node is required to be represented with all its context in the XML document, it should be bound to an XPath expression instead of an XGraph expression. The reason is that since XGraph should describe the structure of the subgraph explicitly, if the result is required to contain the context of each node in an XGraph object, the expression of this XGraph object must include all the nodes as the context of this XGraph object in the schema and be too complex.

For example, query

Query 8 *FOR #G AS* $< a(\%a) < /b/d(\%d) > /c/\%d/\%a >$ *IN document*(*text.xml*)
RETURN #G

will return the result

```
<a id="a1">
<b id="b1">
<d id="d1" a="a">5</d>
</b>
<c id="c1" d="d1">
</a>
```

Topological constraints can also be embedded in GXQuery expression as constraint in XPath

or XGraph expression or in WHERE clause. For example, query

Query 9 *FOR #G AS < a < //e > /c/d > IN document("text.xml")*
LET #SG AS < //b/d > IN document("test.xml")
WHERE #G OVERLAPPING #SG WITH <d>
RETURN #G

returns the result

```
<a id="a1">
<c id="c1">
<d id="d1">5</d>
</c>
<e id="e1">
</a>
<a id="a1">
<c id="c2">
<d id="d1">5</d>
</c>
<e id="e1">
</a>
```

4. Use Cases of GXQuery

In this section, we will present some use cases of GXQuery in order to explain GXQuery furthermore. The data as example is shown in Figure 2. Corresponding graph structure is shown in Figure 1.

1. Retrieves the names of persons who have published papers in both conference and journal

   Solution in GXQuery

   ```
   for $n as bib[journal/paper/author/
   person(%p)]
   /conference/paper/author/%p/name in
   document("bib.xml")
   return $n
   ```

   Expected result:

   ```
   <name>p2</name>
   ```

2. Retrieve the names of persons who published paper in the conference with himself as one of PCmembers.

   Solution in GXQuery:

   ```
   for $i as bib//person(%p)/PCmember
   /conference /paper/author/%p/ name in
   document("bib.xml")
   return $i
   ```

   Expected result:

   ```
   <name>p2</name>
   ```

3. Retrieve the names of conferences which share at least one paper with same title and author in a journal.

   Solution in GXQuery

   ```
   for #G as bib/<conference<name>/
   <paper<title>/author>*> in document
   ("bib.xml")
   for #SG as bib/<journal/<paper<title>/
   author>*>
   in document("bib.xml")
   let $i as conference/name in #G
   where #G overlap #SG with <paper<
   title>/author>
   return $i
   ```

   Expected result:

   ```
   <name>n1</name>
   ```

4. Retrieve the names of persons who have not published papers in the conference with some paper whose author has name n_1.

   Solution in GXQuery

   ```
   for #G as bib//<paper/author/person/
   name> in document("bib.xml")
   for #SG as bib/<conference/ [paper/
   author/person [name="n1"]]/paper
   /author/<person>*> in
   document("bib.xml")
   let $i as paper/author/person/
   name in #G
   where #G DISJOINT #SG
   return $i
   ```

   Expected result:

   ```
   <name>p3</name>
   ```

5. Retrieve persons who have been PCmembers of some conference, the result should contain the name and address of each person and the name of the conference where he has been the PCmember.

   Solution in GXQuery

   ```
   for #G as bib// <person<name>
   <address>/PCmember/<conference/
   name>*> in document("bib.xml")
   return #G
   ```

Expected result:

```
<person>
<name>p2</name>
<address>a1</address>
<PCmember><conference><name>n2
</name></conference></PCmember>
</person>
```

6. Retrieve the conferences with their names and papers. Each paper has title, names and addresses of the authors.

   Solution in GXQuery

```
for #G as bib/<conference
<name[value()="n1"]>/paper<title>/
<author<name>/address>*>
return #G
```

   Expected result:

```
<conference>
<name>n1</name>
<paper>
<title>t1</title>
<author>
<name>p3</name>
<address>a3<name>
</author>
</paper> <title>t2</title>
<author>
<name>p2</name>
<address>a2<name>
</author>
</paper>
</conference>
```

5. Preliminary Techniques for GXQuery Evaluation

In this section, the implementation issues are proposed. At first the framework of the GXQuery evaluation is presented. Then, the operators related to GXQuery are introduced with their implementation.

5.1. The framework of GXQuery evaluation

The basic framework of GXQuery evaluation is to split the query into XGraph expressions. Each XGraph expression is processed separately. Then, the topological constraint is judged. As the last step, the final results are constructed.

For example, Query 9 in Section 3 contains two XGraph expressions. To process Query 9 on the graph shown in Figure 3, as the first step, these two XGraph expressions of <a<//e>/c/d> and <//b/d> are processed respectively. The results of the former XGraph expression is S_1={(a_1, e_1, c_1, d_1), (a_1, e_1, c_2, d_2), (a_1, e_1, c_1, d_2), (a_1, e_2, c_1, d_1), (a_1, e_2, c_2, d_2), (a_1, e_1, c_1, d_2)}. The result of the later XGraph expression is (b_1, d_1). Then the graphs in S_1 are filtered with the graph (b_1, d_1) with the constraint "overlapping with <d>". The results are (a_1, e_1, c_1, d_1) and (a_1, e_2, c_1, d_1). The last step is to construct the graphs with values and tags with the ids for these tuples.

5.2. Implementation of operators

From Section 5.1, two major operators are *graph pattern matching* for XGraph subquery processing and the *topological constraint judgement*. They can be processed with the algorithms based on the labelling scheme introduced in Section 2. The algorithms for graph pattern matching and topological constraint judgement are presented in [13] and [12], respectively.

The basic idea of the former one is to split a graph pattern into bipartite graphs and process each pattern by bipartite graphs in a hash-based method.

The latter is to process topological constraint judgement with existing graph pattern matching algorithms. This algorithm is to build a structure to store the labelling schemes of the nodes in the graph of topological constraints and then use such structure to filter the results as the results.

6. Related Work

Existing query languages for XML [2] are only based on tree structure without considering graph features. Even though Lorel [1] considers IDREF, it considered path as basic unit instead of graph. The disadvantage of using path as the basic unit is that circle and topological relationships are difficult to be represented.

There are also several query languages designed for describing recursion relationship [6] and graph matching [7, 10]. They focus on the description of query in the form of labelled graph without complex structural restrictions and topological restrictions. Query languages

related to RDF [8] can be used to represent query in the form of graph. But current query languages do not consider topological restrictions.

Currently, existing work of querying graph-structured XML mainly focus on structural query of subgraph/subtree matching in a [14, 4]. None of them considers the problem of topological query.

7. Conclusions

In this paper, GXQuery, an extension of XQuery is presented to support represent queries in form of general graph and queries with topological constraints. We present XGraph as an extension of XPath language to express graph matching in flexible forms. GXQuery is compatible with XQuery and XGraph expression can be embedded in XPath and XQuery expressions. GXQuery can represent five topological relationships between XGraph objects. As far as we know, this is the first paper that considers topological query on XML data.

In this paper, only the description of GXQuery is presented. Efficient implementation of GXQuery is a challenging work and it is left for further research.

References

[1] S. ABITEBOUL, D. QUASS, J. MCHUGH, J. WIDOM, J. L. WIENER, The lorel query language for semistructured data. Int. J. on Digital Libraries, 1(1) (1997), pp. 68–88.

[2] A. BONIFATI, S. CERI, Comparative analysis of five xml query languages. *SIGMOD Record*, 29(1) (2000), pp. 68–79.

[3] D. D. CHAMBERLIN, D. FLORESCU, J. ROBIE, Query: A Query language for XML. In *W3C Working Draft*, `http://www.w3.org/TR/xquery`, 2001.

[4] L. CHEN, A. GUPTA, M. E. KURUL, Stackbased algorithms for pattern matching on dags. In *VLDB*, (2005), pp. 493–504.

[5] J. CLARK, S. DEROSE, XML path language (XPath). In *W3C Recommendation, 16 November 1999*, `http://www.w3.org/TR/xpath`, 1999.

[6] I. F. CRUZ, A. O. MENDELZON, P. T. WOOD, A graphical query language supporting recursion. In *SIGMOD Conference*, (1987), pp. 323–330.

[7] R. H. GÜTING, Graphdb: Modeling and querying graphs in databases. In J. B. Bocca, M. Jarke, C. Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, (1994), pp. 297–308. Morgan Kaufmann.

[8] P. HAASE, J. BROEKSTRA, A. EBERHART, R. VOLZ, A comparison of rdf query languages. In *International Semantic Web Conference*, (2004), pp. 502–517.

[9] H. V. J. RAKESH AGRAWAL, A. BORGIDA, Efficient management of transitive relationships in large data and knowledge bases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data (SIGMOD 1989)*, (May 1989), pp. 253–262, Portland, Oregon.

[10] L. SHENG, Z. M. ÖZSOYOGLU, G. ÖZSOYOGLU, A graph query language and its query processing. In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Australia*, IEEE Computer Society, (1999), pp. 572–581.

[11] J. P. TIM BRAY, C. M. SPERBERG-MCQUEEN, F. YERGEAU, Extensible markup language (xml) 1.0 (third edition). In *W3C Recommendation 04 February 2004*, `http://www.w3.org/TR/REC-xml/`, 2004.

[12] H. WANG, J. LI, J. LUO, Topological queries on graph-structured xml data: Models and implementations. *International Journal of Information Technology*, 4(4) (2008), pp. 213–220.

[13] H. WANG, J. LI, J. LUO, H. GAO, Hash-base subgraph query processing method for graph-structured xml documents. *PVLDB*, 1(1) (2008), pp. 478–489.

[14] V. J. T. ZOGRAFOULA VAGENA, M. M. MORO, Twig query processing over graph-structured xml data. In *Proceedings of the Seventh International Workshop on the Web and Databases (WebDB 2004)*, (2004), pp. 43–48.

Received: February, 2007
Revised: April, 2009
Accepted: March, 2011

Contact addresses:

Hongzhi Wang
Department of Computer Science and Technology
Harbin Institute of Technology, Harbin Institute of Technology
Harbin, China
e-mail: `wangzh@hit.edu.cn`

Jianzhong Li
Department of Computer Science and Technology
Harbin Institute of Technology, Harbin Institute of Technology
Harbin, China

HONGZHI WANG is an Associate Professor at Department of Computer Science and Technology, Harbin Institute of Technology, China. His areas of interest include XML and information integration.

JIANZHONG LI is a Professor at Department of Computer Science and Technology, Harbin Institute of Technology, China. His areas of interest include parallel database, sensor network, XML, digital library and compressed database.