International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:12, 2008

Topological Queries on Graph-structured XML Data: Models and Implementations

Hongzhi Wang, Jianzhong Li, and Jizhou Luo

***Abstract*—In many applications, data is in graph structure, which can be naturally represented as graph-structured XML. Existing queries defined on tree-structured and graph-structured XML data mainly focus on subgraph matching, which can not cover all the requirements of querying on graph. In this paper, a new kind of queries, topological query on graph-structured XML is presented. This kind of queries consider not only the structure of subgraph but also the topological relationship between subgraphs. With existing subgraph query processing algorithms, efficient algorithms for topological query processing are designed. Experimental results show the efficiency of implementation algorithms.

***Keywords*—XML, Graph Structure, Topological query.

I. Introduction

In many applications, data can be modeled as a directed graph and graph-structured XML can be naturally used to describe graph-structured data. For example, Figure I shows a graph structure representing relationships between authors and academic papers.

Query processing on graph-structured data brings new challenges. Some query processing techniques have been presented such as [6], [11], [7], [3].However, current techniques for graph-structured data have limitations. An notable one is that current research on querying graph-structured data mainly focus on subgraph matching without considering topological relationship between subgraphs. Topological relationship of subgraphs can be defined as the relationship of subgraphs in position. For example, in Figure 2(a), subgraph $(a1, b1, a2, b2)$ has overlapping relationship with subgraph $(a1, b1, a2, c1)$ in because they share some nodes. In some applications of querying subgraphs on a large graph, topological relationship can be used to describe the constraint of subgraph more subtly.

An example of the application of topological constraints on the graph in Figure I is shown in Example 1.

Example 1:
Retrieve the conferences sharing at least one paper with same title and authors with a journal. Such query use a topological relationship "overlapping" as a constraint. Processing this query in the graph in Figure I is to find subgraphs containing conference and sharing common parts of author information with some subgraph containing information about journal.

Processing topological query is also a challengeable problem. Current techniques of query processing on graph database can be classified into two kinds.

- Match subgraph with structural constraint on a large graph [3], [13]. Such techniques retrieve subgraphs matching a given subgraph. Such techniques can only solve a subproblem of processing topological query. Additionally, in some cases, the result of subgraph query is very large. If such subgraph is just a constraint, only some features of intermediate results are useful. Therefore, it is not efficient for topological query processing.
- Select graphs with some feature in a large set of graphs [14], [15], [8]. For a topological query, the goal is to find some subgraph is a large graph satisfying some constraint. Therefore, such techniques cannot be applied to topological query processing directly.

Current techniques can not be applied directly on topological query processing. For processing topological queries efficiently, we present a series of algorithms based on interval based labeling scheme. Even subgraph matching cannot be avoided during the processing, we only store useful features as intermediate results and filter as many useless nodes as possible.

The contributions of this paper can be summarized as follows:

- Topological query, a novel type of queries on graph, is presented. As we know, this is the first paper of considering topological relationship between subgraphs in query processing on graph data.
- We design a series of efficient algorithms for processing topological queries with various topological constraints. With feature extracting, massive storage of intermediate results is avoided.
- Experimental results show that our methods use a small extra cost of subgraph query processing and have good scalability.

This paper is organized as follows. In Section II, some background knowledge is presented. In Section III, the definitions of topological query and various topological relationships are presented. We design topological query processing algorithms in Section IV. Experimental results are shown in Section V. In Section VI, we give an overview of work related to this paper. We draw the conclusions in Section VII.

II. Preliminaries

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

A. Data Model

With IDREF between two elements representing their reference relationship [12], XML data can be modeled as a labelled

Hongzhi Wang is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, email: wangzh@hit.edu.cn
Jianzhong Li is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, email: lijzh@hit.edu.cn
Jizhou Luo is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, email:luojz@hit.edu.cn

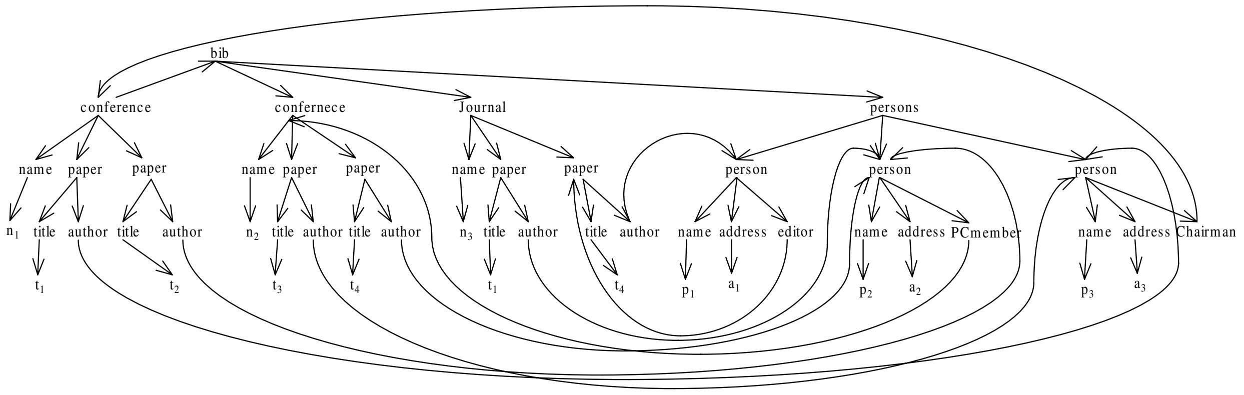


Fig. 1. An Example of Graph Structure

digraph: elements and attributes are mapped into nodes of the graph; directed nesting and reference relationships are mapped into edges in the graph.

B. Reachibility Labeling Scheme for Graph-structured XML Data

The goal of labeling scheme of XML data is to represent the structural relationship so that the relationship between nodes in an XML graph can be judged by their labels quickly. With a suitable labeling scheme, structural query processing can be efficient. In this paper, we focus on *reachability labeling scheme*, which is used to judge the reachability relationship between nodes in an XML graph. We use an extension version of reachability labeling scheme presented in [9]. In this labeling scheme, each node of a graph is assigned a $postid$ and some intervals. For two nodes a and b in a graph, $a \rightsquigarrow b$ if and only if $b.postid$ is contained by one of intervals associated with a, so that the reachability relationship between two nodes in the graph can be judged by their labels without other information.

The steps of encoding a graph G is:

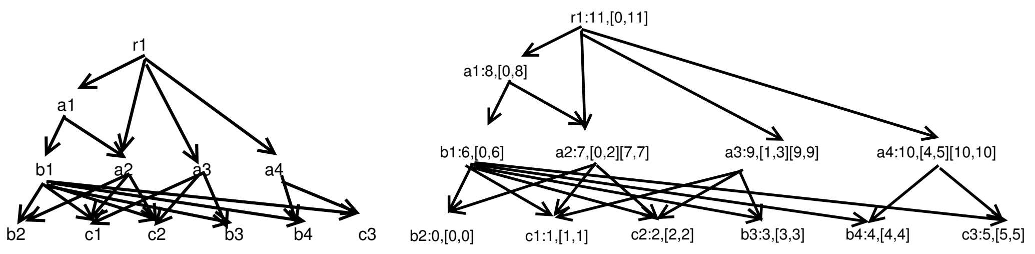
1) A DAG D is generated from G by contracting each strongly connected components (SCC for brief) in G into a node.
2) Find an optimal tree-cover T of the DAG D generated in the first step. An id and an interval is assigned each node in T. For a node n, its id is its postorder. In tts interval, $[x, y]$, y is the postorder of n and x is the smallest postorder number of all n's descendants in T. Note that during the traversal, when an node n_c contracted from a SCC is met, its number of postorder is increased by $number_c$, the number of nodes in this SCC in G instead of 1. Suppose the postorder of the node met before n_c is pc. The id of this node is an series of numbers of $pc+1$, $pc+2$, $\cdots$, $pc+number_c$.
3) Examine all the nodes of D in the reverse topological order. At each node n, copy and merge, if possible, all the intervals of its out-going nodes in D to its code.
4) All nodes in the same SCC in G are assigned to same intervals as the node n_c contracted from this strongly connected graph in D. Each of the $postid$ of n_c in D is assigned to a node in this SCC.

For example, the reachability coding of graph in Figure 2(a) is shown in Figure 2(b). For each node in Figure 2(b), the number after colon is its id and a series of intervals after id are the interval set of this node. The id of $c2$ is contained in interval $[0, 2]$ of $a3$. By such relationship, we can judge that $a3$ reaches $c2$.

III. Concepts of Topological Queries

In this section, we will give the concept of topological query. Intuitively speaking, topological query is to use topological relationship to describe a graph. To retrieve some graphs with some topological constraint with graph in some schema, we present six kinds of topological relationships: "connecting", "connected by", "disjoint", "overlapping", "containing" and "contained by". At first, We give the intuitions of these six relationships. Then we will gives formal definition of these relationships and topological constraints based on them.

- A graph g_1 has "connecting" relationship with another graph g_2 means that there is a path from some node in g_1 to some node in g_2. For example, in Figure 2(a), subgraph ($a1$,$b1$) is connecting subgraph ($a2$,$c2$) because both $a2$ and $b1$ can reach $c2$.
- A graph g_1 has "connected by" relationship with another graph g_2 means that there is a path from some node in g_2 to some node in g_1. For example, in Figure 2(a), subgraph ($a2$,$c2$)is connected by subgraph ($a1$,$b1$).
- A graph g_1 has "disjoint" relationship with another graph g_2 means that no node is shared by g_1 and g_2. For example, in Figure 2(a), subgraphs ($a2$,$c2$)and ($a1$,$b1$) are disjoint since they shares no node.
- A graph g_1 has "overlapping" relationship with another graph g_2 means that g_1 and g_2 shares at least one node. For example, in Figure 2(a), subgraphs ($a2$,$c2$)and ($a3$,$c2$) are overlapping since they shares a node $c2$.
- A graph g_1 has "containing" relationship with another graph g_2 means that g_2 is a subgraph of g_1.
- A graph g_1 has "contained by" relationship with another graph g_2 means g_1 is a subgraph of g_2.



(a) Example of Graph (b) Coding of Example Graph

Fig. 2. An Example of Reachibility Labeling Scheme

The graph structure of XML data can be represented as a labeled, directed graph $G = (V, E)$ with V the set of nodes and E the set of edges. There is a function $tag : V \to TAG$ where TAG is the set of tags and $tag(v)$ is the tag of v. Since topological query is based on subgraph query, we will define subgraph query at first, then define topological relations. Based on these concepts, topological query is defined formally.

Definition 1 (Subgraph Query): A graph $Q = (V, E)$ is a subgraph query, where there are two functions $tag : V \to TAG$ and $rel : E \to AXIS$. TAG is the set of tags and $AXIS$ is the set of node relations such as parent-child(PC for brief) or ancestor-descendant(AD for brief).

In Definition 1, the set of AXIS describes the relationship between two nodes. Based on the definition of XPath [4], there are 13 kinds of relationship. Here we only discuss two axis in common use, PC and AD. In a graph G, two nodes n_1 and n_2 $\in V_G$ has *PC relationship* if there is a edge $(n_1, n_2) \in E_G$. In a graph G, two nodes n_1 and $n_2 \in V_G$ has AD relationship if there is a path from n_1 to n_2 in G.

Definition 2 (Result of Subgraph Query): The result of a subgraph query $Q = (V_Q, E_Q)$ over a graph G is

$\mathbb{R}_{G,Q} = \{g|\exists$ a equivalent function $f_g : V_g \to V_Q$ and a total and single valued function $p_g : V_g \to V_G$, such that $\forall n \in V_g, tag(n) = tag(f_g(n)) = tag(p_g(n))$and $\forall e = (n_1, n_2) \in E_g, p_g(n_1), p_g(n_2)$ satisfy the relationship $rel(f(n_1), f(n_2))$and if $n_1 \neq n_2$ then $f_g(n_1) \neq f_g(n_2)$, $p_g(n_1) \neq p_g(n_2).\}$. $g \in \mathbb{R}_{G,Q}$ is represented as $g \models_G Q$.

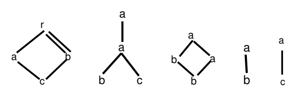
An example of subgraph query is shown in Example 2.

Example 2: A subgraph query on data in Figure 2(a) is shown in Figure 3(a). The direction of each edge is from top to bottom. Single line represents PC-relationship and double line represents AD-relationship. This query represents a subgraph with an r node as source. This a node must have PC relationship with an a node and AD relationship with a b node. Such a node and b node must have PC relationship between the same c node. The results of this subgraph query include ($r1$, $a2$, $b1$, $c1$), ($r1$, $a2$, $b1$, $c2$), ($r1$, $a3$, $b1$, $c1$),($r1$, $a3$, $b1$, $c2$) and ($r1$, $a4$, $b1$, $c3$).

Definition 3 ("Connecting" Relationship): Two labeled, directed graphs $g_1 \models_G Q_1$ has "Connecting" relationship with $g_2 \models_G Q_2$ in graph G if $\exists n_1 \in V_{g_1} \exists n_2 \in V_{g_2}$, in graph G there is a path from $p_{g_1}(n_1)$ to $p_{g_2}(n_2)$.

Definition 4 ("Connected" Relationship): Two labeled, directed graphs $g_1 \models_G Q_1$ has "Connected by" relationship with $g_2 \models_G Q_2$ in graph G if $\exists n_1 \in V_{g_1} \exists n_2 \in V_{g_2}$, in graph G there is a path from $p_{g_2}(n_2)$ to $p_{g_1}(n_1)$.

Definition 5 (Disjoint Relationship): Two labeled, directed graphs $g_1 \models_G Q_1$ has "Disjoint" relationship with $g_2 \models_G Q_2$ in graph G if $\forall n_1 \in V_{g_1} \forall n_2 \in V_{g_2}$, $p_{g_1}(n_1) \neq p_{g_2}(n_2)$.



(a) (b) (c) (d) (e)

Fig. 3. Example Queries

Definition 6 (Overlapping Relationship): Two labeled, directed graphs $g_1 \models_G Q_1$ has "Overlapping" relationship with $g_2 \models_G Q_2$ in graph G if $\exists n_1 \in V_{g_1} \exists n_2 \in V_{g_2}$, $p_{g_1}(n_1) = p_{g_2}(n_2)$.

Definition 7 (Containing Relationship): Two labeled, directed graphs $g_1 \models_G Q_1$ has "containing" relationship with $g_2 \models_G Q_2$ in graph G if $\forall n_2 \in V_{g_2} \exists n_1 \in V_{g_1}$, $p_{g_1}(n_1) = p_{g_2}(n_2)$.

Definition 8 (Contained by Relationship): Two labeled, directed graphs $g_1 \models_G Q_1$ has "Contained by" relationship with $g_2 \models_G Q_2$ in graph G if $\forall n_1 \in V_{g_1} \exists n_2 \in V_{g_2}$, $p_{g_1}(n_1) = p_{g_2}(n_2)$.

"g_1 has topological relationships r with g_2" can be represented as $g \preceq_{G,r} g_2$

Definition 9 (Topological Query): A topological query Q is a triple $(G_1, G_2, relation)$, where G_1 and G_2 are subgraph queries and $relation$ is one of "connecting", "connected by", "disjoint", "overlapping", "containing" and "contained by".

Definition 10: The result of topological query Q=(G_1, G_2, *relationship*) over a graph G is

$\mathbb{R}_{G,Q} = \{g | g \models_G G_1 \wedge \exists g_2 (g_2 \in G \wedge g_2 \models_G G_2 \wedge g \preceq_{G,relationship} g_2)\}$, where type is some topological relationship type.

An example in Example 3 is to illustrate the concept of topological result.

Example 3: A topological query $Q = (G_1, G_2, \text{"Overlapping"})$ with G_1 in Figure 3(a) and G_2 in Figure 3(b). Q is to retrieve the subgraphs matching G_1 and have at least one node in some subgraph matching G_2. The results include ($r1$, $a2$, $b1$, $c1$), ($r1$, $a2$, $b1$, $c2$), ($r1$, $a3$, $b1$, $c1$) and ($r1$, $a3$, $b1$, $c2$). ($r1$, $a4$, $b1$, $c3$) is not a result of Q. It is because that it has none node overlapping any subgraph matching G_2.

IV. Evaluation of Topological Queries

In this section, we present algorithms for the evaluation of topological queries. We find that topological relationship can be classified into two classes, query-related and query-unrelated. Query-related relationship means that just from the query the relationship between the results can be judged. Such relationships includes "Containing" and "Contained by". For another four topological relationships, the relationship between subgraphs cannot be judged just from the query directly but have to be judged from the results of subgraph queries. A topological query with query-related relationship are called a *query-related query*. Otherwise, it is called a *query-unrelated query*.

For a query-related query, the subgraph query G can be generated from the topological query Q and the result of Q can be obtained directly from the result of G. To process query-unrelated query, we design an uniform framework with different filters for deferent relationship.

In this section, at first, the strategy of processing query-related queries is presented. Then we will introduce the algorithms for processing query-unrelated queries.

This paper focuses on topological query processing. As their sub-queries, the processing of subgraph query is beyond the scope of this paper. Existing subgraph query processing algorithms [3], [13] are used as a function.

A. Processing Query-Related Topological Queries

In this subsection, we present the processing strategy for query-related topological queries.

Definition 11 (Query-Related Relationship): A topological relationship r is a query-related relationship if for a query-related query $Q = (G_1, G_2, r)$, the topological relationship r between two subgraphs matching G_1 and G_2 can be judged directly from the query G_1 and G_2 without generating the result.

The topological relationships of "Containing" and "Contained by" can be judged directly from the query. It is because for two subgraph queries G_1 and G_2 on G, if G_2 is a subgraph of G_1, any result g_1 of G_1 must contain a result of G_2. Otherwise, if some g_1 does not contain any result of G_2 there must be some node in G_2 that can not be matched by any node of g_1. Then the corresponding node in G_1 can not be matched by any node of g_1. Then g_1 is not a result of G_1. Therefore, just from the query, "Containing" and "Contained by" relationship can be judged from the query.

Based on the feature of query-related relationship, a topological query $Q = (G_1, G_2, type)$ with relationship "Containing" can be process with Algorithm 1.

Algorithm 1 Query$_{containing}$(G, Q)

```
check whether $Q.G_2$ is a subgraph of $Q.G_1$
if $Q.G_2$ is a subgraph of $Q.G_1$ then
  return QuerySubgraph($G$, $Q.G_1$)
else
  return $\phi$
```

In this algorithm, if $Q.G_2$ is a subgraph of $Q.G_1$, any result of $Q.G_1$ must contain a subgraph matching $Q.G_2$. Otherwise, any result of $Q.G_1$ does not contain a result matching $Q.G_2$.

A topological query $Q = (G_1, G_2, type)$ with relationship "Contained by" can be process with Algorithm 2.

In Algorithm 2, checking whether $Q.G_1$ is a subgraph of $Q.G_2$ can use a subgraph matching algorithm and all the

Algorithm 2 Query$_{contained}$(G, Q)

```
check whether $Q.G_1$ is a subgraph of $Q.G_2$
if $Q.G_2$ is a subgraph of $Q.G_1$ then
  $\mathbb{G}$ =QuerySubgraph($G$, $Q.G_2$)
  for each $g$ in $\mathbb{G}$ do
    $\mathbb{S}'$=subgraps of g matching $Q.G_1$
    $\mathbb{S} = \mathbb{S} \bigcup \mathbb{S}'$
  return $\mathbb{S}$
else
  return $\phi$
```

subgraphs matching $Q.G_1$ in $Q.G_2$ are found and each is given an identify. Note that one node in $Q.G_2$ may be attached multiple identifies. The step obtaining subgraph of g matching $Q.G_1$ is different from subgraph matching. It is because during the step of checking whether $Q.G_1$ is a subgraph of $Q.G_2$, the nodes of $Q.G_1$ related to $Q.G_2$ is identified and during projection, nodes corresponding to the query node with the same identify can be extracted directly.

We use an example in Example 4 to show the processing of "Contained by" topological query.

Example 4: For a topological query $Q = (G_1$, G_2, "Contained by") on graph in Figure 2(a) with G_1 in Figure 3(d) and G_2 in Figure 3(c). At first, the relationship between G_1 and G_2 is checked and G_1 is a subgraph of G_2 and there are two subgraph matching G_1 in G_2: the left-top one is defined as group1 and the right-bottom one is defined as group2. After processing subgraph query G_2, a result ($a1$,$b1$,$a2$,$b2$) is obtained with ($a1$,$b1$) matching group1 and ($a2$,$b2$) matching group2. Thus,($a1$,$b1$) and ($a2$,$b2$) are two results of Q.

Complexity Analysis Since this paper focuses on topological query processing, during analysis, we take subgraph query processing algorithm as an oracle and use $Cost_{M(q,G)}$ to represent the cost of processing subgraph q on graph G. Based on Algorithm 4, the time complexity of "Containing" topological query processing only depends on the time of subgraph query and is at most $Cost_{M(G,Q.G_1)}$ + $Cost_{M(Q.G_1,Q.G_2)}$. Based on Algorithm 5, the time complexity of "Contained by" is $Cost_{M(G,Q.G_2)}$ + $Cost_{M(Q.G_2,Q.G_1)}$ + $O(|Q.G_1.V| * result_{size})$, where $result_{size}$ is the number of final results. It is because besides the cost of graph matching, extra cost is the generation of subgraphs matching $Q.Q_1$ from the subgraphs matching $Q.Q_2$ and with identity, such cost is just the cost of generating all final results.

B. Framework of Processing Query-unrelated Topological Queries

In this subsection, we will present the framework of topological query evaluation. The evaluation strategies for 4 Query-unrelated topological queries share the same framework.

The framework can be describe in Algorithm 3.

Algorithm 3 Query(G, Q)

```
$\mathbb{G}_{filter}$ = QuerySubgraph($G$, $Q.G_2$)
$S_{filter}$=GenerateFilters($\mathbb{G}_{filter}$, $Q.type$)
$\mathbb{G}_{result}$ = QuerySubgraphWithFilter($G$, $Q.G_1$,$S_{filter}$)
return $\mathbb{G}_{result}$
```

In the framework, for a graph G and a topological query $Q = (G_1, G_2, type)$, at first, G_2 is processed and then from the results of G_2, the set of filters are generated. At last, subgraph query G_1 is processed, during the processing of G_1, the filters generated in the last step is applied to filter the results not satisfying the topological constraint.

Note that in order to avoid storing intermediate results, same filters can be generated when one result matching G_2 and such result is not necessary to be stored.

The benefit of sharing the same framework is that a topological query with the logical combination of multiple topological constraints can be processed with the combination of filters.

C. Algorithms of Processing Query-unrelated Topological Queries

In this subsection, we will present algorithms of processing query-unrelated topological queries based on the framework presented in Section IV-B.

Connecting

For processing a topological query with topological relationship "connecting", reachibility labeling scheme is used. As introduced in Section II-B, each node is attached to an id and a list of intervals. For a topological query $Q = (G_1, G_2, \text{“}\textit{connecting}\text{”})$, a sorted list L is maintained as filter. When a subgraph g_2 matching G_2 is generated, id of each node g is inserted to L. When a subgraph g_1 matching G_1 is generated, all intervals of g_1's nodes are used to validate g_1. If an interval $[x, y]$ contains some id in L, it means some node of g_1 can reach a node in some subgraph matching G_2. Then g_1 is a result of Q. The algorithm is sketched in Algorithm 4.

Algorithm 4 Query$_{connecting}(G,\ Q)$
```
$\mathbb{G}_{filter}$ = QuerySubgraph($G$, $Q.G_2$)
for each $g \in \mathbb{G}_{filter}$ do
  for each node $n \in g$ do
    insert $n.id$ to $L$
$\mathbb{G}_{par}$ = QuerySubgraph($G$, $Q.G_1$)
for each graph $gin\mathbb{G}_{par}$ do
  for each node $n \in g$ do
    for each interval $[x, y]$ in $n.intervals$ do
      if $\exists id \in L, x \leq id \leq y$ then
        add $g$ to $\mathbb{G}_{result}$ and stop the processing of current $g$
return $\mathbb{G}_{result}$
```

To accelerate the inserting and searching, L is implemented with binary search tree with insert and delete cost $o(logn)$, where n is the number of nodes.

Complexity Analysis The run time of connecting algorithm can be estimated as following. $Cost = Cost_{M(G,Q.G_1)} + Cost_{M(G,Q.G_2)} + cost_{insert_L} \cdot |Q.G_2.V| \cdot result_(M(G,Q.G_2)) + cost_{search_L} \cdot result_{interval}(M(G,Q.G_1))$, where $result\ (M(G,Q.G_2))$ and

$result_{interval}(M(G,Q.G_1)$ are the number of results of $M(G,Q.G_2)$ and the total number of intervals in the result of $M(G,Q.G_1)$, respectively. Since the size of L is at most $|G.V|$, $cost_{insert_L}$ and $cost_{search_L}$ are both at most $o(log|G.V|)$.

Connected by Processing a topological query $Q(G_1, G_2, \text{“}\textit{Connected by}\text{”})$ uses intervals of nodes in all the results of G_2. All the intervals are maintained in an ordered list L. When a result g of G_1 is generated, L is searched for id of each node in g to find the interval $[x, y]$ satisfying $x \leq id \leq y$. If such interval can be found, it means that some node in g is connected by some node in a subgraph as a result of G_2 and g is accepted as a result. The pseudo code of processing "Connected by" topological query is shown in Algorithm 5.

Algorithm 5 Query$_{connected}(G,\ Q)$
```
$\mathbb{G}_{filter}$ = QuerySubgraph($G$, $Q.G_2$)
for each $g \in \mathbb{G}_{filter}$ do
  for each node $n$ in $g$ do
    insert all the interval in $n.intervals$ to $L$
$\mathbb{G}_{par}$ = QuerySubgraph($G$, $Q.G_1$)
for each graph $gin\mathbb{G}_{par}$ do
  for each node $n \in g$ do
    if $\exists [x, y] \in L, x \leq n.id \leq y$ then
      add $g$ to $\mathbb{G}_{result}$ and stop the processing of current $g$
return $\mathbb{G}_{result}$
```

For efficient inserting and searching, L is in order of x value and maintained with a binary search tree. In order to reduce the space cost of L, overlapping and nested intervals can be merged. Since the judgement condition is existing a interval containing id, the merge of overlapping and nested intervals will not affect the result. The merging is performed online. When an interval $[x, y]$ is inserted into L, the merging strategies are shown as follows.

- If an interval $[a, b]$ in L is found satisfying $a \leq x \leq y \leq b$, then $[x, y]$ will not be inserted.
- If an interval $[a, b]$ in L is found satisfying $x \leq a \leq b \leq y$, then $[x, y]$ is inserted to L and $[a, b]$ is delete from L.
- If two intervals $[a, b]$ and $[c, d]$ in L are found satisfying $x-1 \leq b$ and $c \leq y+1$, then $[a, b]$ and $[c, d]$ are replaced by $[a, d]$.
- If an interval $[a, b]$ in L is found satisfying $a \leq x \leq b+1 \leq y$, then $[a, b]$ is replaced by $[a, y]$.
- If an interval $[a, b]$ in L is found satisfying $x \leq a \leq y+1 \leq b$, then $[a, b]$ is replaced by $[x, b]$.

With merging strategies, all intervals in L are not overlapping or nesting. When a result g of G_1 is generated, L is searched for id of each node in g to find the interval $[x, y]$ with largest x value among the intervals with x value smaller than id. If such interval can be found and id satisfies $x \leq id \leq y$, g is considered as a result.

An example in Example 5 is used to illustrate the processing of a "Connected by" query.

Example 5: For a topological graph $Q = (G_1, G_2$, "Contained by"), where G_1 is the subgraph query in Figure 3(d) and G_2 is the subgraph query in Figure 3(e). At the first step, subgraph query G_2 is processed and then results ($a2$, $c1$), ($a2$, $c2$), ($a3$, $c1$), ($a3$, $c2$) and ($a4$, $c3$) are returned. Then intervals of each nodes in result are added to the filter. For $a2$, $[0, 2]$ and $[7, 7]$ are added to filter set L. For $c1$ and $c2$, no additional interval is added to L since their intervals $[1, 1]$ and $[2, 2]$ are contained in existing interval in L, $[0, 2]$. When $a3$ is considered, interval $[1, 3]$ is merged to $[0, 2]$ in L and $[0, 3]$ takes the place of $[0, 2]$ in Lr. For $a4$, $[4, 5]$ is merged with $[0, 3]$ and $[0, 5]$ takes the place of $[0, 3]$ in L. $[10, 10]$ is also required to merged with $[9, 9]$. After processing all nodes of subgraphs matching G_2, intervals in L include $[0,5]$,$[7,7]$ and $[9,10]$. During the process of G_1, when a subgraph matching G_1 such as ($a1$,$b1$) is generated, for each id of its nodes, L is searched. For ($a1$,$b1$), no interval in L contains any of the ids of $a1$ and $b1$. So ($a1$,$b1$) is not a result for Q. For ($a2$,$b1$), since the id of $b1$ is contained in the interval [0,5], it is a result of Q.

Complexity Analysis The run time of connecting algorithm can be estimated as $Cost = Cost_{M(G,Q.G_1)} + Cost_{M(G,Q.G_2)} + cost_{insert_L} \cdot |Q.G_2.V| \cdot result_{interval}(M(G,Q.G_2)) + cost_{search_L} \cdot |Q.G_1.V| \cdot result(M(G,Q.G_1))$, where $result_{interval}(M(G,Q.G_2))$ and

$result(M(G,Q.G_1))$ is the total number of intervals in the results of $M(G,Q.G_2)$ and number of results of $M(G,Q.G_1)$, respectively. With our merging strategy and the property of the labeling scheme, the number of intervals are at most $|G.V|$. Therefore, $cost_{insert_L}$ and $cost_{search_L}$ are both at most $o(log|G.V|)$.

Overlapping and Disjoint

The filter for a topological query $Q = (G_1, G_2, type)$ with $type$ overlapping or disjoint is a set S of ids of nodes in the subgraphs matching G_2. The difference of processing these two types of topological queries is the filter condition. For a subgraph g matching G_1, the filter condition of "Overlapping" topological query is that for some node n in g, a id equalling to $n.id$ in S can be found. While the filter condition of "Disjoint" topological query is that $n.id$ can not be found in S for all nodes n in g. The description of the algorithm to process overlapping is in Algorithm 6. The algorithm for disjoint is similar except during the filtering step, the graph g with ids of all nodes not in S are considered as a final result.

Algorithm 6 Query$_{Overlapping}$(G, Q)

```
$\mathbb{G}_{filter}$ = QuerySubgraph($G$, $Q.G_2$)
for each $g \in \mathbb{G}_{filter}$ do
  for each node $n \in g$ do
    insert $n.id$ to $S$
$\mathbb{G}_{par}$ = QuerySubgraph($G$, $Q.G_1$)
for each graph $gin\mathbb{G}_{par}$ do
  for each node $n \in g$ do
    if $n.id$ can be found in $S$ then
      add $g$ to $\mathbb{G}_{result}$ and stop the processing of current $g$
return $\mathbb{G}_{result}$
```

For efficient inserting and searching, S can be implemented with a hash set. **Complexity Analysis** The run time of overlapping and disjoint algorithms are similar and can be estimated as following. $Cost = Cost_{M(G,Q.G_1)} + Cost_{M(G,Q.G_2)} + cost_{inserts_S} \cdot |Q.G_2.V| \cdot result(M(G,Q.G_2)) + cost_{searchs_S} \cdot |Q.G_1.V| \cdot result(M(G,Q.G_1))$, where $result(M(G,Q.G_2))$ and $result(M(G,Q.G_1))$ are the number of results of $M(G,Q.G_2)$ and number of results of $M(G,Q.G_1)$, respectively. With hash set as data structure, $cost_{inserts_S}$ and $cost_{searchs_S}$ are both $o(1)$.

V. Experimental Results

In this section, we present the experimental results and analysis of our algorithms for processing topological queries.

TABLE I
Statistics of the XMark Datasets

factor	0.1	0.2	0.3	0.4	0.5
Size	11.3M	22.8M	34.0M	45.3M	56.2M
#Nodes	175382	351241	524067	697342	870628
#Edges	206129	413110	616228	820437	1024072

A. Experimental Setup

All of our experiments were performed on a PC with Pentium 2.4GMHz, 512M memory and 40G IDE hard disk. The OS is WindowsXP Professional. We implemented our system using Microsoft Visual C++ 6.0. We implemented all our algorithms in this paper. We implemented subgraph query processing algorithms presented in[13] as the subgraph query processing module and embedded the filter generation and result filtering in result generation step of subgraph query processing module. The subgraph query processing algorithm in [13] is a disk-based algorithm. we fix block size 1K and buffer size 32K.

The dataset we tested is the XMark benchmark dataset [10]. It can be modeled as a graph and has a fairly complicated schema, with circles as subgraphs and many nodes with multiple parents. We choose factor 0.1, 0.2, 0.3, 0.4, 0.5 to generate XMark documents. Their parameters are shown in Table I

For topological queries, we choose three groups of graphs, groupU1: (G_{u11},G_{u12}), groupU2: (G_{u21},G_{u22}) and groupU3: (G_{u31},G_{u32})for "Connecting", "Connected", "Overlapping" and "Disjoint" and three groups of graphs groupR1: (G_{r11},G_{r12}), groupR2: (G_{r21},G_{r22}) and groupR3: (G_{r31},G_{r32}) for "Containing" and "Contained by". The graphs are shown in Figure V, where double line represents AD-relationship between nodes. G_{u32} and G_{r21} are the same graphs. For each group for query-unrelated queries, we design four queries on them with four topological relationship, respectively. In each group, the former subgraph is used as "G_1" and the latter subgraph as "G_2" in the form of topological query $Q = (G_1, G_2, type)$.

For each group for query-related queries with subgraph query G_1 and G_2, we design two queries $(G_1, G_2, \text{"Containing"})$ and $(G_2, G_1, \text{"Contained by"})$ on them. Since the case that G_1 does not contain G_2 can only processed by comparing the two queries and the case is trivial, in each group, G_2 is a subgraph of G_1.

B. Comparison

In this subsection, we will present the comparison results between our algorithms and graph matching. For query-unrelated query, We compare the processing time of query-unrelated queries with the sum of processing time of the two subgraph queries in the topological query. The results is shown in Figure 5(a). Since the three group of subgraph queries for query-related queries all include one subgraph containing another, we also compare the processing time of query-related queries with the larger one of two subgraph queries in each group. The results are shown in Figure 5(b). From the results, it can be seen that comparing with the time of processing

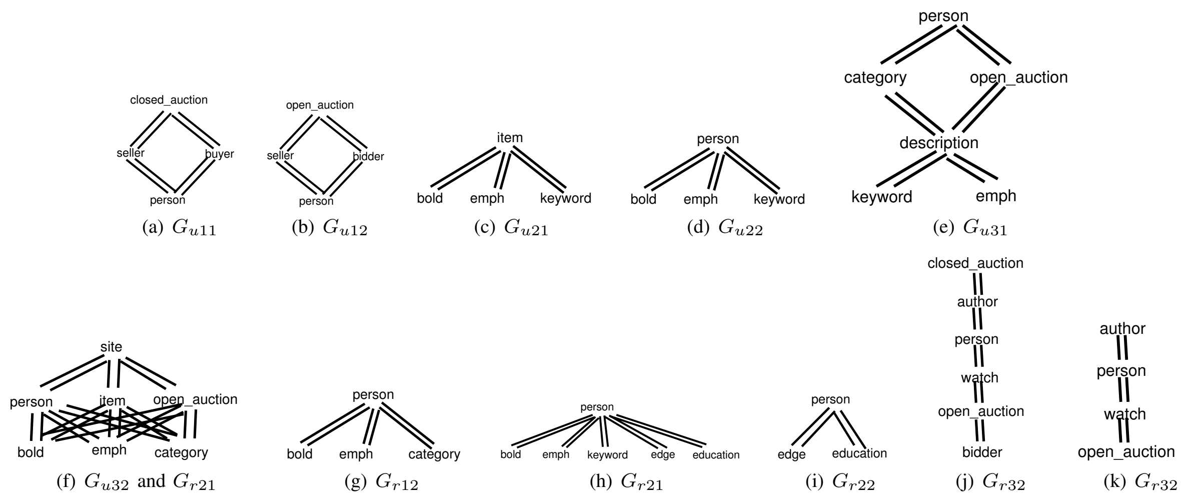


Fig. 4. Query Graphs

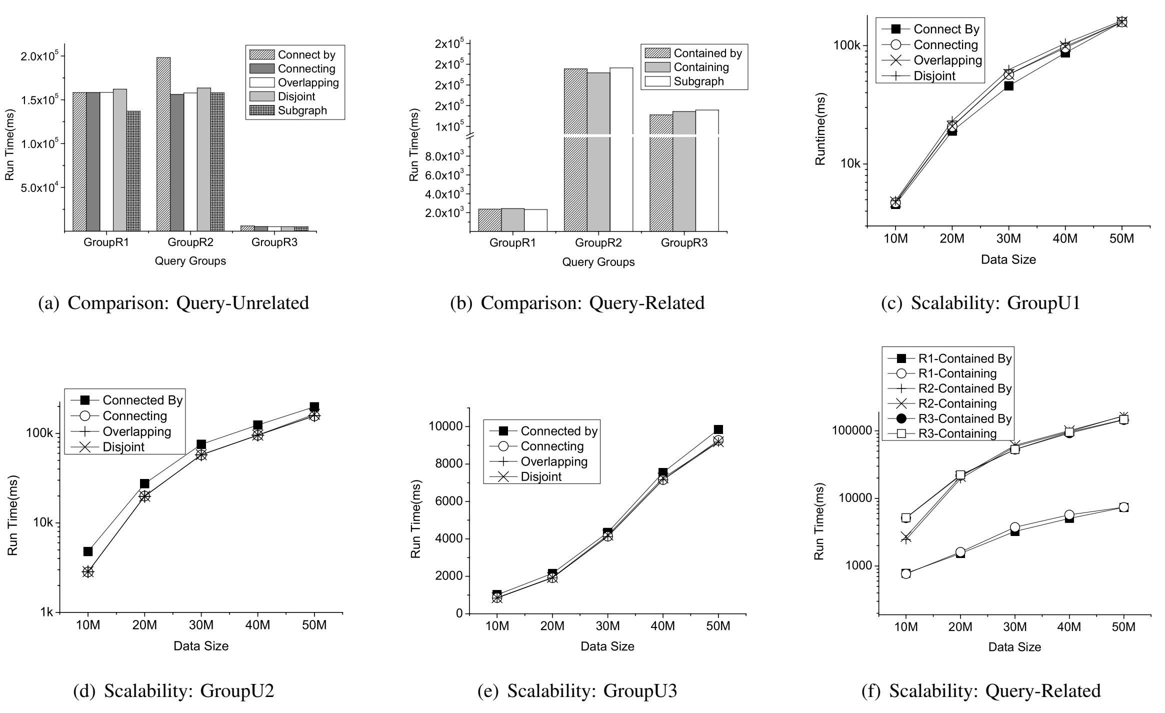


Fig. 5. Experimental Results

subgraph queries, the extra costs of our algorithms are very small.

C. Scalability

In this subsection, we will present the results of scalability of our algorithms. We range XMark data from 10M to 50M. The results of query-unrelated queries are shown in Figure 5(c), Figure 5(d) and Figure 5(e). The results of query-related queries are shown in Figure 5(f). Note that the run time of Figure 5(c), Figure 5(e) and Figure 5(f) are in log scale. From the results, the run time of our algorithm increases a little faster than linearly but slower than exponentially with data size. It is because that our algorithm increases with the size of partial results of the subgraph query in topological constraint and final results. The size of partial results may increase faster than linearly because the numbers of nodes matching each query node in subgraph query increases about linearly with the size of XML document and so that the in some cases the increase of the number of results may be the production of the increase time of nodes matching every query node.

VI. Related Work

In this section, we give an overview of previous work related to this paper. Related work can be classified into two kinds. One is the model and representation of query on graph, the other is query processing algorithms on graph.

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 [5] and graph matching [6], [11]. 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 [7] 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 graph [16], [3], [13]. None of them considers the problem of topological query processing.

Another kind of work on querying graph is to retrieve graph satisfying some condition from a large set of graphs. Such work includes [14], [15], [8]. Such works consider the structural features of a graph. However, none of them considers the topological features of graph.

VII. Conclusions

In this paper, topological query, a new kind of query for graph-structured XML data, is presented. This query is to retrieve subgraph with some given topological relationship with some other subgraph in graph-structured data. We define six types of topological relationship between subgraphs to be used as the topological constraint. We give the definitions and evaluation algorithms of topological queries. Experimental results show that our implementation algorithms use a small extra cost of subgraph query and scales with the size of results of subgraph query in it.

Future work includes efficient index building and holistic algorithms to process topological queries.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[2] A. Bonifati and S. Ceri. Comparative analysis of five xml query languages. *SIGMOD Record*, 29(1):68–79, 2000.

[3] L. Chen, A. Gupta, and M. E. Kurul. Stack-based algorithms for pattern matching on dags. In *VLDB*, pages 493–504, 2005.

[4] J. Clark and S. DeRose. XML path language (XPath). In *W3C Recommendation, 16 November 1999*, http://www.w3.org/TR/xpath, 1999.

[5] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *SIGMOD Conference*, pages 323–330, 1987.

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

[7] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *International Semantic Web Conference*, pages 502–517, 2004.

[8] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, page 38, 2006.

[9] H. V. J. Rakesh Agrawal, Alexander 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)*, pages 253–262, Portland, Oregon, May 1989.

[10] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A benchmark for XML data management. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB 2002)*, pages 974–985, 2002.

[11] L. Sheng, Z. M. Özsoyoglu, and 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, Austrialia*, pages 572–581. IEEE Computer Society, 1999.

[12] J. P. Tim Bray, C. M. Sperberg-McQueen, and F. Yergeau. Extensible markup language (xml) 1.0 (third edition). In *W3C Recommendation 04 February 2004*, http://www.w3.org/TR/REC-xml/, 2004.

[13] H. Wang, W. Wang, X. Lin, and J. Li. Subgraph join: Efficient processing subgraph queries on graph-structured xml document. In *WAIM*, pages 68–80, 2005.

[14] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD Conference*, pages 335–346, 2004.

[15] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD Conference*, pages 766–777, 2005.

[16] V. J. T. Zografoula Vagena, Mirella Moura Moro. Twig query processing over graph-structured xml data. In *Proceedings of the Seventh International Workshop on the Web and Databases(WebDB 2004)*, pages 43–48, 2004.

Hongzhi Wang Hongzhi Wang received his PHD in computer science from Harbin Institute of Technology in 2008. He is a lecturer of Department of Computer Science and Technology, Harbin Institute of Technology. His research area is XML data management and information integration.

Jianzhong Li Jianzhong Li received his PHD in computer science from Harbin Institute of Technology in 1985. He is a professor of Department of Computer Science and Technology, Harbin Institute of Technology. His research area includes parallel database, sensor network, data mining, data warehouse, compressed database and XML data management.

Jizhou Luo Jizhou Luo received his PHD in computer science from Harbin Institute of Technology in 2006. He is a lecturer of Department of Computer Science and Technology, Harbin Institute of Technology. His research area is compressed database.