Fast Leaf-to-Root Holistic Twig Query on XML Spatiotemporal Data

Luyi Bai, Yin Li, Jiemin Liu*
College of Information Science and Engineering, Northeastern University, Shenyang 110819, China.

*Corresponding author. Tel.: +8618903340011; email: liujm@neuq.edu.cn
Manuscript submitted March 20, 2016; accepted July 17, 2016.
doi: 10.17706/jcp.12.6.534-542

Abstract: With the increasing applications based on location, the researches on spatiotemporal data, especially queries of spatiotemporal data have attracted a lot of attention. XML, as a standard language of information exchanging over the Web, should have the ability to query spatiotemporal data. In this paper, we propose an algorithm, TwigSPFast, for matching a spatiotemporal XML query twig pattern. We represent spatiotemporal data by adding spatial and temporal attributes in common data and extend Dewey code to mark spatiotemporal data for special process and determine structure relationship of spatiotemporal nodes. Our technique uses streams to store leaf nodes in XML corresponding to leaf query node and filters the streams to delete unmatched nodes, after filtering, we build output list for every matched leaf node that matches the query path from leaf node to root. It can be proved that TwigSPFast is optimal in time complexity and space complexity.

Keywords: Spatiotemporal data, query, Dewey code, XML twig pattern.

1. Introduction

With the prompt development of positioning system, a considerable amount of spatiotemporal applications [1]-[3] merged, e.g. environment management, land management, and geographic information system. Spatiotemporal data is characterized by both spatial [4], [5] and temporal [6], [7] semantics. Development and research in this area started decades ago, when management and manipulation of data, relating to both spatial and temporal changes, was recognized as an indispensable assignment. In the development of spatiotemporal data, several spatiotemporal data models have been proposed [8]-[10], e.g. the snapshot model, the space-time composite data model, object-relationship model, and spatiotemporal object-oriented data model. With the development of spatiotemporal data, people begin to study the management of spatiotemporal XML data whose query operation is a key part including query, index, retrieval and topological relations [10]-[13].

Currently, a large number of theories have been proposed for XML queries [14]-[18], such as global matching theory, two element structure connection theory, and approximate query processing theory. Also some researchers concentrate on querying temporal data in XML [19], [20], but the research about spatiotemporal querying is little [21]. The process of querying is based on a good data model that can present spatial and temporal attributes of spatiotemporal data. Bai et al. [11] proposed a data model which use a 5-tuple to represent spatiotemporal data. It added information of position, motion, and time in the traditional XML data to describe the spatiotemporal attributes. An important work for querying is to
determine the structure of nodes in XML. To solve this problem, lots of code scheme have been proposed [14], [22], [23]. The algorithms proposed in [14], [22] need to read and filter lots of data and push data down to stack continually what waste lots of time and space. Extended Dewey code [15] is a kind of prefix code that not only can determine structure relationship of nodes in XML but also can get ancestor nodes according the Dewey code of node. The work of querying spatiotemporal data in XML is very arduous and it leave lots of interspace for us to research.

In this paper, we use a 3-tuple \((ATTR, PS, TM)\) to represent spatiotemporal data without changing tree structure of XML document. We use minimum bounding rectangle to depict the position attribute, which can represent point, line, region in space well. The time attribute is represented by a time period consist of two points. We also extended Dewey coding by adding an integer at the start of code to mark spatiotemporal node for special process. In the process of querying, we associate every leaf query node with a stream to store matched leaf nodes in XML and filter the leaf nodes in stream to delete nodes unmatched, after that, we sort the leaf query nodes to determine the sequence of processing. In the phase of getting desired results, we call the algorithm buildList to build output list from leaf node to root that matches query path.

The rest of paper is organized as follows. Section 2 discusses preliminaries and related work. Section 3 focuses on spatiotemporal XML twig pattern matching algorithm called TwigSP Fast and then presents the analysis of the correctness and complexity of time and space. Section 4 concludes this paper.

2. Preliminaries

2.1. Twig Pattern Matching

**Definition 1** (twig query matching)  
Given a twig query \(Q = (V_Q, E_Q)\) and a XML database \(D = (V_D, E_D)\), the process of matching \(Q\) in \(D\) is a mapping from query node \(Q\) to element node \(D\), the mapping \(h: \{u: u \in Q\} \rightarrow \{x: x \in D\}\) must satisfy the following condition:

(i) Query node predicates are satisfied by the corresponding database nodes, and they have the same tag name.

(ii) The structure relationships between query nodes are satisfied by the corresponding database nodes.

2.2. Semantic Foundations of Spatiotemporal Data

Spatiotemporal data have a set of characteristics that make them distinctly different from the more familiar lists and tables of alphanumeric data used in traditional business applications, the data model is restricted to tree structure in XML document especially. In [11], Bai et al. measures fuzzy spatiotemporal XML data by five dimensions: \(OID, ATTR, FP, FT\) and \(FM\). In the case of spatiotemporal data, we measure it by three dimensions according to characteristics of spatiotemporal data on the basis of those five dimensions. The dimensions of spatiotemporal data are shown as follows:

**ATTR**: It is used to describe static properties of spatiotemporal data (e.g., land owner’s name, area, population, etc.). There may be one or more attributes in a spatiotemporal data which dimension heavily depends on the application domain.

**PT**: It describes the spatial attributes of spatiotemporal data, which contains point, line and region.

**TM**: It describes the temporal attribute of spatiotemporal data, which contains time point and time interval.

2.3. Extended Dewey Code

Dewey code is a kind of prefix encoding, it can only determine the structure relationship between nodes, when it comes to get node’s name from root to a certain node, the Dewey code need extending that is
proposed in [4].

There is a label \( t \) in XML, \( CT(t) = \{ t_0, t_1, \ldots, t_{n-1} \} \) is used to express all child nodes of \( t \) got from DTD structure information of XML document. For every element \( e_i \) tagged \( t_i \), it can be given a integer \( x_i \) satisfied \( x_i \mod n = i \). Extended Dewey code can be computed by depth first traversal and code of every XML node is expressed by integer vector. We use label \((u)\) to express the code of node \( u \), if label \((u)=\) label \((s)\), \( x \) then \( s \) is the parent node of \( u \), the value of \( x \) can be computed as follows:

If \( u \) is a text node, then \( x = -1 \)

Else, suppose \( u \) is the kth \((k = 0, 1, 2, \ldots, n-1)\) tag of node \( t_x \), then

\( (2.1) \) if \( u \) is the leftmost child, then \( x = k \);

\( (2.2) \) else, suppose the code of \( u \)'s left child node is label \((s)\), \( y \), if \( (y \mod n) < k \), then \( x = \left[ \frac{y}{n} \right] \times n + k \); else \( x = \left[ \frac{y}{n} \right] \times n + k \). \( n \) is number of \( t_x \)'s child nodes.

![Diagram](image)

Fig. 1. Examples of extended Dewey code.

We can translate the code into sequence of element’s name from root to a certain node by a finite state transducer \((FST)\) after giving extended Dewey code of one node. \( FST \) is expressed by a 5-tuple \((I, S, i, \delta, \tau)\):

(i) \( I = \mathbb{Z} \cup \{-1\} \), \( \mathbb{Z} \) is nonnegative integer set;
(ii) \( S = \Sigma \cup \{PCDATA\} \), PCDATA expresses the state of text;
(iii) initial state of \( i \) is element’s name of root in XML document;
(iv) the function of state translating is defined as follows: for \( \forall s \in \Sigma \), if \( x=-1 \), then \( \delta(t, x) = \) PCDATA; else \( \delta(t, x) = t_{ki}, k = x \mod n \);
(v) \( \tau \) depicts name of current state. The example of extended Dewey code and \( FST \) is showed in Fig. 1.

3. **Holistic Twig Query on Spatiotemporal Data**

The process of querying is based on a good data model, to depict spatiotemporal attributes well, we simplify the data model proposed in [1] to represent spatiotemporal data with minimum bounding rectangle by adding spatial and temporal attributes in general data.
3.1. Process Spatiotemporal Node in XML Tree

The spatial and temporal attribute of spatiotemporal data makes it distinctly different from common data, the same process of common node is not available when it comes to spatiotemporal node. As for spatiotemporal node, we need to mark and process it specially. The extended Dewey introduced in section 2 is not available for spatiotemporal data, we need to do some improvement: we add an integer that value 0 or 1 at the start of the code sequence, 0 represents that the node is a common node and 1 represents the node is a spatiotemporal node. For spatiotemporal node at the start of the code sequence, 0 represents that the node is a common node. conversely, the first integer of node 0, it depicts the node owner is a common node. For spatiotemporal node, we need to mark and process it specially. The extended Dewey introduced in section 2 is not available when it comes to spatiotemporal node. As for the same process of common node is not available when it comes to spatiotemporal node. As for

![XML document tree marked spatiotemporal node.](image)

3.2. Notations

Let $Q$ denote twig query pattern, $P_f$ denotes the path pattern from root of $Q$ to node $f (\in Q)$. In algorithm TwigSPFast, we use the following twig node operations on query node $q$: leafNodes$(q)$ that returns set of leaf query nodes in sub twig query pattern rooted $q$, isSP$(q)$ that determines whether node $q$ is a spatiotemporal node, parent$(q)$ that returns parent query nodes of $q$ and isLeafNode$(q)$ that determines whether node $q$ is a leaf node. Algorithm TwigSPFast associates every leaf query node $f$ in twig query pattern with a stream named $T_f$ and associates every query node $s$ with a list named $L_s$, elements in $L_s$ are expressed with a 2-tuple (element $e$, an array of pointers). The operations on $T_f$ is: eof$(T_f)$ that determines whether the pointer of stream $T_f$ is end, advance$(T_f)$ that moves the pointer of stream to the next node, delete$(T_f)$ that removes the current element the pointer points and getElement$(T_f)$ that returns the current element the pointer of $T_f$ points. The operations on $L_s$ is: addtoList$(L_s, \text{element } a, \text{pointers})$ that adds 2-tuple(element $a$, pointers) to the list $L_s$ and addPointer$(L_p, \text{tuple}_p, \text{pointer to tuple}_f)$ that adds a pointer of tuple_f to a pointer array of 2-tuple tuple_p in $L_p$.

3.3. TwigSPFast Algorithm
Algorithm TwigSPFast, which computes answers to a query path pattern on spatiotemporal data, is presented as follows. It's a holistic twig query on spatiotemporal data without changing the tree structure of XML.

**Algorithm TwigSPFast**

01: for each $f \in \text{leafNodes}(\text{root})$
02: \hspace{1em} locateMatchedLabel($f$)
03: SortNodeArray[] = DBLSort(TopBranchingNode)
04: $i = 0$
05: while ($\neg \text{end}(\text{root})$ and $i<\text{SortNodeArray.length}$)
06: \hspace{1em} $f = \text{SortNodeArray}[i]$
07: \hspace{1em} while ($\neg \text{eof}(\mathcal{T}_f)$)
08: \hspace{2em} buildList($f, \text{getElement}(\mathcal{T}_f), \text{parent}(f)$)
09: \hspace{2em} advance($\mathcal{T}_f$)
10: \hspace{1em} $i++$
11: outputSolutions()

**Procedure locateMatchedLabel($f$)**

/* Assume that the path from the root to element $\text{getElement}(\mathcal{T}_f)$ is $n_1/n_2/.../n_k$ and $P_f$ denotes the path pattern from the root to leaf node $f$ */

01: while ($\neg \text{eof}(\mathcal{T}_f)$)
02: \hspace{1em} if (($n_1/n_2/.../n_k$ matches pattern $P_f$) $\land$ fitness($n_k, f$))
03: \hspace{2em} delete($\mathcal{T}_f$)
04: \hspace{2em} else
05: \hspace{2em} advance($\mathcal{T}_f$)
06: \hspace{2em} reset the pointer of $\mathcal{T}_f$ to the first element of $\mathcal{T}_f$

**Function fitness($e, f$)**

01: if (isSP($f$))
02: \hspace{1em} if ($f.x_0 < e.x_0 < e.x_1 < f.x_1$) $\land$ ($f.y_0 < e.y_0 < e.y_1 < f.y_1$) $\land$
03: \hspace{2em} $(f.t_0 < e.t_0 < e.t_1 < f.t_1)$
04: \hspace{2em} return true
05: \hspace{2em} else if ($e == f$)
06: \hspace{2em} return true
07: \hspace{2em} else return false

Algorithm TwigSPFast is a single phase algorithm, it calls the algorithm locateMatchedLabel to filter the stream $\mathcal{T}_f$ of leaf node $f$ (line 1-2). Also the algorithm locateMatchedLabel delete the spatiotemporal node that doesn’t match the query condition and filter the element that doesn’t match path pattern $P_f$. The algorithm call DBL(TopBranchingNode) to get array of sorted leaf node of TopBranchingNode (line 3). Line 6 get the processed leaf query node $f$ and establish output list for element in $\mathcal{T}_f$ at line 7-9.

DBL sort is very important, it determines the process order of leaf query node. DBL($n$) returns all branch nodes $b$ and leaf nodes $f$ of sub twig query tree rooted $n$, $b$ and $f$ must satisfy: there is no other branch nodes in the path from root $n$ to $b$ and there is no other branch nodes in the path from root $n$ to $f$. There are a twig query $Q$ and a XML database $D$, suppose query node $n \in Q$, we associate each node $n$ with a variable
nodeCount to count the amount of XML nodes matching n. If n is a leaf node, nodeCount is the amount of element in $T_f$, else, nodeCount values the minimum nodeCount of all query nodes in DBL(n). The detail of DBL sort is showed as follow:

```
Algorithm DBLSort(n)
/* At the beginning, the value nodeCount(n) for each query node n is zero */
01: if (isLeafNode(n))
02:   locateMatchedLabel(n)
03:   nodeCount[n] = $T_n$.length
04: s = $L_{DBL}(n)$
05: for each node a in list s
06:    DBLSort(a)
07:    if (nodeCount[n] < nodeCount[a])
08:       nodeCount[n] = nodeCount[a]
09: sort(s, nodeCount[a])  // ascendantly
```

In the algorithm TwigSPFast, line 8 calls buildList algorithm to establish list of each query node. Algorithm buildList is the core part of the whole algorithm, lines 1-10 establish the list of leaf query node, line 11 establishes list of internal query node by calling algorithm buildInternalList.

```
Algorithm buildList(f, e, p)
/* let s be a set of element b that is an ancestor of e such that b can be match node p in the path solution of e to path pattern $P_f$ */
01: if (p != null)
02:   for each b ∈ s
03:     if (tuple_p(b, pointers array_p) ∈ $L_p$ ∧ $P_p$ matches $P_f$)
```

In the algorithm TwigSPFast, if $f$ is the first node of SortNodeArray and there is not 2-tuple ($e$, null) in list $L_f$, add 2-tuple ($e$, null) to $L_f$ (line 1-3); if $f$ is not the first node of SortNodeArray (line 4), the list $L_a$ of closest common ancestor of $f$ and previous query node of $f$ in SortNodeArray that has contained the 2-tuple($element e_2$, pointers) and the tuple and its ancestors match the path query $P_f$ correspond to $e$ (lines 5-6), if there is no tuple($e$, null), then add tuple ($e$, null) to $L_f$ (lines 7-8).
3.4 Analysis of Query Algorithm

3.4.1 Analysis of correctness

**Theorem 1:** Given a twig query $Q$ and a XML document $D$, TwigSPFast can return all matching results in $D$ correctly.

**Proof:** Suppose twig query $Q$ is $(A_1, A_2, \ldots, A_n)$ rooted at $A_1$. Let $\{A_k, A_{k+1}, \ldots, A_n\} \mid 1 \leq k \leq n \}$ be the set of leaf nodes. Then, $A_j \mid 1 \leq j \leq n \}$ is topBranchingNode. When $(a_1, a_2, \ldots, a_n)$ matches $(A_1, A_2, \ldots, A_n)$, $a_i \mid i=k, k+1, \ldots, n \}$ must be in $T_{A_i}$ after filtering of $T_f$. In lines 1-2 of TwigSPFast, then in lines 5-10, the algorithm gets next leaf query node $f$ according to results of DBL sort and call buildList to build lists for all elements of $T_f$, so, $(a_1, a_2, \ldots, a_n)$ is all in $L_{A_i}$, according to the process of build list. At last, the algorithm call function outputSolutions to output query result $(a_1, a_2, \ldots, a_n)$ matching $(A_1, A_2, \ldots, A_n)$. When algorithm TwigSPFast output $(a_1, a_2, \ldots, a_n)$, it indicates $a_i \mid i=1, 2, \ldots, n \}$ is in the list $L_{A_i}$, we can know that $a_j \mid j \text{ is must in the path from } A_1 \text{ to } A_j$, according to the process of build list. So, $(a_1, a_2, \ldots, a_n)$ matches $(A_1, A_2, \ldots, A_n)$.

3.4.2 Analysis of time complexity

**Theorem 2:** Given a twig query $Q$ and a XML document $D$, the temporal complexity in the worst case is $O(n_1 d)$. $n_1$ is the sum of leaf nodes in $T_f$, $n_2$ is the sum of leaf nodes in $T_f$ after filtering of locateMatchedLabel, $d$ is the maximum between depth of $Q$ and depth of $D$.

**Proof:** In algorithm TwigSPFast, lines 1-2 filter all leaf query nodes in $T_f$, time complexity is $O(n_1 d)$, lines 5-10 build list by calling buildList for all leaf query nodes in $T_f$, time complexity is $O(n_2 d)$, $n_1 > n_2$, so, the time complexity of the whole algorithm is $O(n_1 d)$.

Above all, we can know that the algorithm TwigSPFast is optimal in time complexity.

3.4.3 Analysis of space complexity

**Theorem 3:** Given a twig query $Q$ and a XML document $D$, the temporal complexity in the worst case is $O(n_1+n_2 d)$. $n_1$ is the sum of leaf nodes in $T_f$, $n_2$ is the sum of leaf nodes in $T_f$ after filtering of locateMatchedLabel, $d$ is the depth of $D$.

**Proof:** In algorithm TwigSPFast, cost of space is mainly consist of stream and list. The space complexity of all leaf nodes of twig query $Q$ in $T_f$ is $O(n_1)$. Lines 5-10 build list by calling buildList for all leaf query nodes in $T_f$, the algorithm establish a new tuple for elements matched for every query node in the worst case, the space complexity of list is $O(n_2 d)$. So, the space complexity of the whole algorithm is $O(n_1+n_2 d)$.

Above all, we can know that the algorithm TwigSPFast is optimal in space complexity.

4. Conclusions

In this paper we developed a 3-tuple($\text{ATTR, PS, TM}$) to represent spatiotemporal data in XML as a triangle, which can represent spatiotemporal attributes well without changing the tree structure of XML. Based on
this data model, an effective algorithm to match the desired twigs is proposed after extending Dewey code to mark spatiotemporal. The algorithm TwigSPFast reduces cost of time and space by sorting leaf query nodes and filtering leaf nodes that don't match query path in XML, so, algorithm TwigSPFast is optimal in time complexity and space complexity.

The next work we will do is to query fuzzy spatiotemporal XML data based on a good spatiotemporal XML data model and improve the efficiency of query algorithm.

Acknowledgments

The authors wish to thank the anonymous referees for their valuable comments and suggestions, which improved the technical content and the presentation of the paper. The work was supported by the National Natural Science Foundation of China (No. 61402087), the Natural Science Foundation of Hebei Province (No. F2015501049), the Scientific Research Fund of Hebei Education Department (No. QN2014339), and the Technology Support Project of NEU (No. XNK 2015003).

References


**Luyi Bai** received his PhD degree from Northeastern University, China. He is currently an associate professor at Northeastern University at Qinhuangdao, China. His current research interests include uncertain databases and fuzzy spatiotemporal XML data management. He has published papers in several journals such as Integrated Computer-Aided Engineering, Applied Intelligence, Applied Artificial Intelligence, and The International Arab Journal of Information Technology. He is also a member of CCF.

**Yin Li** was born in 1990. Since 2014, he has been a master of science candidate in computer technology from the Northeastern University, China. His current research interests include fuzzy spatiotemporal XML data management.

**Jiemin Liu** received his PhD degree from Northeastern University, China. He is currently a professor and a dean at Northeastern University at Qinhuangdao, China. His current research interests include next generation network and information management system. He has published several papers and academic monographs. He is also a vice chairman of ACM China Council at Qinhuangdao and a chairman of CCF at Qinhuangdao.