Querying Spatiotemporal Data Based on XML Twig Pattern

Luyi Bai, Yin Li, Jiemin Liu*

College of Information Science and Engineering, Northeastern University, Shenyang 110819, China.

* Corresponding author. Tel.: 18903340011; email: liujm@neuq.edu.cn
Manuscript submitted July 10, 2016; accepted August 23, 2016.
doi: 10.17706/jsw.11.12.1199-1206

Abstract: With the increasing applications based on location, 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, has the ability to query spatiotemporal data. In this paper, we propose an algorithm, TwigStackSP, for matching a spatiotemporal XML query twig pattern. We represent spatiotemporal data by adding spatial and temporal attributes in general data and extend region coding scheme to filter the nodes in P-C relationship. Our technique uses a chain of linked stacks to compactly represent partial results to root-to-leaf query paths, which are then composed to obtain matches for the twig pattern. It can be proved that TwigStackSP is I/O and CPU optimal when there are no text nodes in P-C relationship of twig pattern.

Keywords: Spatiotemporal data, query, region encoding scheme, XML twig pattern.

1. Introduction

With the prompt development of positioning system, a considerable amount of spatiotemporal applications merged [1], [2], e.g. environment management and geographic information system. Spatiotemporal data is characterized by both spatial [3] and temporal [4] semantics. With the development of spatiotemporal data, some operations based on spatiotemporal data are researched, such as spatiotemporal data query [5] and spatiotemporal data indexing [6].

Currently, a large number of theories have been proposed for XML queries [7], [8], such as *global matching theory*, *two element structure connection theory*, and *approximate query processing theory*. Most of the researches in this field are based on XML twig pattern [7], [9], [10]. proposed an algorithm called TwigStack that uses a chain of linked stacks to compactly represent partial results of individual query root-to-leaf paths, the approach is I/O and CPU optimal with only ancestor-descendant edges, but for queries with parent-child edges, TwigStack can't control the size of intermediate results. In [9], Lu et al. proposed a new holistic twig join algorithm, called TwigStackList, which is significantly more efficient than TwigStack for queryies with the presence of parent-child edges. while, TwigStackList needs to read more elements from all non-leaf node streams. So, an algorithm, called TJFast, is proposed in [10] based on extended Dewey labeling scheme which only scans query leaf nodes.

With the development of querying general data becomes mature gradually, more and more researchers have found the potential of querying spatiotemporal data in XML [11], [12]. In [11], Chen and Revesz

introduced a Layer Algebra, which is a novel extension of relational algebra for XML and provides the basis of query evaluation and optimization for XML queries. They present a rule-based XML query language, DataFox (Datalog For XML), which combines the simplicity of Datalog with the support for spatiotemporal data in constraint databases. In [12], Bai et al. make fuzzy-set-based extensions to XQuery including truth degrees, fuzzy spatiotemporal linguistic terms, and FLOWR expressions after proposing the architecture of querying fuzzy spatiotemporal data using XQuery. These researches provide a reference for further research in the field of spatiotemporal XML data querying.

In this paper, we simplify the data model proposed in [13] to represent spatiotemporal data with minimum bounding rectangle by adding spatial and temporal attributes in general data. We also extend the region coding by adding childrenClue information which can remove the nodes with P-C relationship that are not in the final results definitely in the first phase of our algorithm TwigStackSP, and we improve the function of getting solutions to make it suitable for processing spatiotemporal XML data.

The rest of paper is organized as follows. Section 2 discusses preliminaries. Section 3 focuses on spatiotemporal XML twig pattern matching algorithm called TwigStackSP and then presents the analysis of the correctness and complexity of the algorithm. 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: uϵQ} $\rightarrow$ {x: $x \in D$} must satisfy the following condition:

- Query node predicates are satisfied by the corresponding database nodes, and they have the same tag name.
- The structure relationships between query nodes are satisfied by the corresponding database nodes.

2.2. Solution

Definition 2 (solution) For any two nodes p, c in subtree rooted q, the node q has a solution when the structure of p and c (Edge(p, c)) is same with the structure of nodes corresponding to their current elements C_p and C_c (Edge(C_p, C_c)). For specifically, the node q has a solution when node q is a leaf node and C_q is not null.

2.3. Extended Region Code

The intermediate results can be reduced for improving query efficiency in processing twig query with P-C relationship by removing some data elements that are not be involved in the final results definitely, if there is enough information, especially the information of sub-element, in XML data encoding. The schema information of XML document will be used when add sub-element information in XML data elements encoding.

Children information of elements is presented with some binary bits named childrenClue. Representing all the different sub-label of label t in XML schema information with DCT(t) = {t_0,t_1, ... ,t_n}, representing all different tag names corresponding to sub-element of XML document data element e with DCE(e) = {t_{e1},t_{e2},...,$t_{e(m-1)}$}, representing all different tag names corresponding to sub-node in P-C relationship of node n in a twig query with DCQ(n) = {t_{n0},t_{n1},...,$t_{n(l-1)}$}. Tag names {t_0,t_1, ... ,t_n} in DCT(t) are sorted by alphabet. Data elements in XML document and childrenClue information of nodes in twig query are represented with n+1 binary bits (b_0, b_1, b_2 ,..., b_n), childrenClue information is computed as follows:

(1) There is no childrenClue information of leaf elements and string elements in XML documents, and

there is also no childrenClue information of leaf nodes and string nodes in twig query.

(2) The first binary bit in childrenClue represent whether elements(nodes) contains text, the first bit in childrenClue of e will be 1, if there are text elements in sub-elements of a data element e, otherwise, it's 0; the first bit in childrenClue of q is 1, if there are text nodes in sub-nodes of a query node q's P-C relationship.

(3) childrenClue (b_0, b_1, b_2 ,..., b_n) of element e in XML document whose tag name is t, if $t_i \in DCT(t)$ and $t_i \in DCE(e)$, $b_{i+1} = 1$, otherwise, $b_{i+1} = 0$.

(4) childrenClue (b_0, b_1, b_2 ,..., b_n) of node q in twig query whose tag name is t, if $t_i \in DCT(t)$ and $t_i \in DCQ(n)$, $b_{i+1} = 1$, otherwise, $b_{i+1} = 0$.

In extended region code, if data element is leaf node or text node, its code is represented by a 3-tuple (start, end, level), other data elements code are represented by a 4-tuple (start, end, level, childrenClue), start, end and level have the same meaning with them in region code respectively, childrenClue is depicted as above.

Supposing that there is same tag name of data element e in XML document and node n in twig query, then we can determine whether data element e is not belong to the final result of twig query according to the result of label(e).childrenClue AND label(n).childrenClue. If the result doesn't equal to label (n).childrenClue, it can be seen that element e mustn't be in the final result. Because there must be a t belongs to DCQ (n), but don't belong to DCE (e), it's also to say, there is a P-C relationship isn't satisfied.

3. Twig Join Algorithms

3.1 Notation

Let q (with or without subscripts) denotes twig patterns, as well as (interchangeably) the root node of the twig pattern. In our algorithms, we make use of the following twig node operations: isLeaf: Node → Bool, isRoot: Node → Bool, isSP: Node → Bool, parent: Node → Node, children: Node → {Node}, PCchildren: Node → {Node}, PCNodes: q → {Node}, childrenClue: Node → childrenClue, and subtreeNodes: Node → {Node}. Children (n) returns all the children nodes of node n in twig query. PCchildren (n) returns the children nodes of node n that have P-C relationship with n in twig query. The result of PCNodes (q) is subnodes that have P-C relationship in sub tree rooted q. childrenClue(n) returns the children clue information childrenClue of node n, and subtreeNodes(n) returns nodes contained in the query tree rooted node n.

In the process of querying, associated with each node q in a query twig pattern there is a stream T_q. What stored in the stream T_q is ordered list of extended region code corresponding to all data elements of query node q in XML document. The nodes in the stream are sorted by their (start) values. The operations over streams are: current (T_q), advance (T_q), eof (T_q), and remove (T_q, e). current(T_q) returns elements of T_n in processing; advance(T_q) move the current element of T_n back; eof(T_q) judges whether the current element is the last one; remove(T_q, e) remove e from T_q. Current (T_q).start, current (T_q).end, current (T_q).level and current(T_q).childrenClue represent the attribute of current element in T_q respectively.

In our stack-based algorithms, we also associate each query node q with a stack S_q. Each data node in the stack consists of a pair: (extended region code of element in T_q, pointer to a node in $S_{parent(q)}$). The operations over stack S_q are: empty, pop, push, topstart, topEnd. The last two operations return start and end information of elements at the top of the stack. At every point during the computation, (i) the nodes in stack S_q (from bottom to top) are guaranteed to lie on a root-to-leaf path in the XML database and sorted by start values, (ii) the set of stacks contain partial and total answers to the query twig pattern.

3.2 TwigStackSP

Before processing twig pattern matching, we should do some changes on the extended region code mentioned in section 2 to be suitable for processing general data firstly, when the data has spatial and temporal attributes. We need to identify whether the data is spatiotemporal because the process between spatiotemporal and general data is different. So we add a bit in childrenClue (b_0, b_1, b_2 ,..., b_n, b_{n+1}) to identify whether the data is spatiotemporal as follow: childrenClue (b_0, b_1, b_2 ,..., b_n, b_{n+1}) of element e in XML document whose tag name is t, if there are sub-elements with spatiotemporal information, b_{n+1} = 1, otherwise, b_{n+1} = 0.

Algorithm TwigStackSP, which computes answers to a query path pattern on spatiotemporal data, is presented in Fig 3. It improved the algorithm TwigStackBE to query spatiotemporal data on the basis of not changing the tree structure of XML.

```
    ALGORITHM TwigstackSP(q)
 // Phase 1
01 cleanStreamSP(q)
//Phase 2
02 While (¬end (q))
03      n_act = getNext (root)
04      If(¬isRoot(n_act))
05          cleanStack(parent(n_act),C_nact →start)
06      if(isRoot(n_act)∨¬empty(S_parent(n_act)))
07          cleanStack(n_act,C_nact →end)
08          moveStreamToStack(T_nact, S_nact, pointertotop(S_parent(n_act)))
09          if(isLeaf(n_act))
10              showSolutions(S_nact, 1)
11              pop(S_nact)
12      else
13          advance(T_nact)
//Phase 3
14 mergeAllPathSolutions()

FUNCTION cleanStreamSP(q)
01      for each n_i in PCNodes(q)
02          if(n_i.childrenClue is NULL )
03              for each e_j in T_ni
04              if((e_j.childrenClue is NULL) OR
(e_j.childrenClue&n_i.childrenClue ≠ n_i.childrenClue))
05                  remove(T_ni, e_j);
FUNCTION cleanStackSP(S_n, actValue)
01      while(¬empty(S_n) ∧topEnd(S_n)<actValue)
02          pop(S_n)
03      end while
```

Algorithm TwigStackSP operates in three phases. In the first phase remove some data elements that are not sure to be involved in the final results in processing twig query with P-C relationship (line 1). In the second phase, some (but not all) solutions to individual query root-to-leaf paths are computed (lines 2-13). In the third phase, these solutions are merge-joined to compute the answers to the query twig pattern (line 14).

In the processing of TwigStackSP query, the function getNext(q) is also very important. The correctness of getNext(q) directly determines whether all occurrences of a twig pattern can be found.

```
FUNCTION getNext(q)
01    If(isLeaf(q))
02        If(q.childrenClue.last = 0)
03            return q;
04        else if((Cq.position ⊆ q.position) ∧(Cq.time⊆q.time))
05            return q;
06    for( each  ni  in children(q))
07        gi  = getNext(ni);
08        If(gi  ≠ ni) return  gi;
09    nmax  = getNmax(q);
10    nmin  = getNmin(q);
11    While(Cq →end <Cnmax →start)
12        advance(Tq);
13    if(Cq →start <Cni →start)
14        return q;
15    else
16        return  nmin;

FUNCTION getNmax(q)
01    startAtt = max{Cni →start |  ni ∈  children(q)}
02    for each  ni  in children(q)
03        if(Cni →start = startAtt)
04            return  ni;

FUNCTION getNmin(q)
01     startAtt = min{Cni →start |  ni ∈  children(q)}
02    for each  ni  in children(q)
03        if(Cni →start = startAtt)
04            return  ni;
```

getNext accesses leftmost node through recursive call from root, and check bottom-to-top to search high level query node with solution. Give a node q, if its children nodes all have solutions, we need to guarantee that q also has solution for returning node q. If there is no element that can be the common ancestor of all elements $C_{q_{ni}}$ in the stream T_q, then return the node with minimum start value.

The key difference between TwigStackSP and TwigStack [3] is that the algorithm TwigStackSP processes querying on spatiotemporal data on the basis of not changing the structure of XML. In this paper, we first extended region code to identify the data with spatiotemporal attributes using a binary bit. In the process of querying, we improve the function getNext to confirm whether the node with spatiotemporal attribute has solution. If a node with spatiotemporal attribute is query node, it need to satisfy the condition as follows:

(i) $q.x_0 \le C_q.x_0 \le C_q.x_1 \le q.x_1$

(ii) $q.y_0 \le C_q.y_0 \le C_q.y_1 \le q.y_1$

(iii) $C_q.t_0 \le q..t_0 \le q.t_1 \le C_q.t_1$

3.3 Analysis of TwigStackSP

In this section we discuss algorithm TwigStackSP for processing twig query, and then we analyze its correctness and complexity.

Lemma 3.1 The data element e removed from stream T_q in the process of cleanStreams in first phase of algorithm TwigStackSP is not in the final result definitely.

Proof: Suppose that data element e removed in the process of cleanStreams is in the final result of twig query, the element e and query node q have the same tag name t, at the same time, e.childrenClue & q.childrenClue ≠q.childrenClue and they are not null because element e is removed. We can know there must be a tag name $t_i \in$ DCQ(q), but $t_i \notin$ DCE(e). It is to say that the query node q has a sub node q_{t_i} labeled t_i and there are P-C relationship between q and q_{t_i}. But element e has no sub node labeled t_i leading to not satisfy the relationship between q and q_{t_i}. It's contradictory with what we supposed, so, the lemma is proved.

Theorem 3.1 Given a query twig pattern q, and an XML database D, Algorithm TwigStackSP correctly returns all answers for q on D.

Proof: In Algorithm TwigStackSP, we first call function cleanStream to remove the nodes in P-C relationship that are not in final results definitely. It can be proved that there are not nodes removed in the final results. Then we repeatedly find getNext (q) for query q. Assume that getNext(q) = q_N. Let A_{q_N} be the set of nodes in the query that are ancestors of q_N. getNext returns all elements from the stream of spatiotemporal nodes in A_{q_N} that are part of a solution that uses h_{q_N}(the first element in T_{q_N} that participates in a solution for the sub-query rooted at q_N). If $q \neq q_N$, in line 5 we pop from parent(q_N)'s stack all elements that are guaranteed not to participate in any new solution. After that, in line 6 we test whether h_{q_N} participate in a solution. We know that q_N has a descendant extension [3]. If $q \neq q_N$ and parent(q_N)'s stack is empty, nodes q_N does not have an ancestor extension. Therefor it is guaranteed not to participate in any solution, so we advance q_N in line 13 and continue with the next iteration. Otherwise, node q_N has both ancestor and descendant extensions and therefor it participates in at least one solution. We then cleanq_N's stack and after that we push h_{q_N} to it (lines 7-8). The pointer to the top of parent (q_N)'s stack correctly identifies all solutions using h_{q_N}. Finally, if h_{q_N} is a leaf node, we decompress the stored solutions from the stacks (lines 9-11).

We use a binary bit to identify whether the node has spatiotemporal attribute depicted with minimum bounding rectangle. We can remove some intermediate result by judging whether there are same bit mark in q and nodes in T_q. Also, in the process of getting solutions, we need to do some special treatment when we process spatiotemporal node. The function cleanStream (q) removed the results that are not sure in the final result except text node in P-C relationship. Then the matching path in the latter process will not be useless result caused by not satisfying P-C relationship. We can conclude that there is only A-D relationship in the latter query process if there is no text node in P-C relationship. The P-C relationship without text node is translated into A-D relationship by the first phase process. That's to say that the CPU and I/O cost is optimal in algorithm TwigStackSP when it process the twig query without text nodes.

4. Conclusion

In this paper we represent spatiotemporal data with minimum bounding rectangle by adding spatial and temporal attributes in general data. On this platform, an effective algorithm to match the desired twigs is proposed after extending the region codding scheme. The extended coding scheme adds childrenClue information to determine whether a node has spatiotemporal attributes. In addition, we can remove the nodes that are not in final results definitely. In addition, the proposed algorithm TwigStackSP is I/O and CPU optimal for a large class of querying spatiotemporal data since we remove the nodes that do not satisfy P-C relationship and get suitable solutions with spatiotemporal attributes by inclusion relationship of spatiotemporal range in getNext (q).

Acknowledgement

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

[1] Deng, S. S., Xia, L. H., & Wang, F. (2009). Analysis of spatio-temporal characteristics of urban land cover and its landscape pattern: A case study in NanHai district of Foshan city. *Proceedings of 2009 Joint IEEE* (pp. 1-9).

[2] Pfoser, D., & Tryfona, N. (1998). Requirements, definitions, and notations for spatiotemporal application environments. *Proceedings of the 6th ACM International Symposium on Advances in Geographic Information System* (pp. 124-130).

[3] Balbiani, P., & Condotta, J. F. (2002). Spatial reasoning about points in a multidimensional setting. *Applied Intelligence*, *17(3)*, 221-238.

[4] Obeid, N. (2005). A formalism for representing and reasoning with temporal information. *Applied Intelligence*, *23(2)*, 109-119.

[5] Sözer, A., Yazici, A., & Oğuztüzün, H, *et al*., Modeling and querying fuzzy spatiotemporal databases. *Information Sciences*, *178(19)*, 3665-3682.

[6] Emrich, T., Kriegel, H. P., & Mamoulis, N., *et al.* (2012). Indexing uncertain spatio-temporal data. *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (pp. 395-404).

[7] Bruno, N., Koudas, N., & Srivastava, D. (2002). Holistic twig joins: optimal XML pattern matching. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (pp. 310-321).

[8] Liu, G., Yao, M., & Wang, D., *et al*. (2011). A novel three-phase XML twig pattern matching algorithm based on version tree. *Proceedings of 8th IEEE Conference on Fuzzy Systems and Knowledge Discovery* (pp. 1678-1688).

[9] Lu, J. H., Chen, T., & Ling, T. W. (2005). Efficient processing of XML twig patterns with parent child edges: A look-ahead approach. *Proceedings of the 13th internal conference on very Large Data Bases* (pp. 193-204).

[10] Lu, J. H., Chen, T., & Ling, T. W. (2005). From region encoding to extended dewey: on efficient processing of XML twig pattern matching. *Proceeding of the 31st ACM Internal Conference on very Large Data Bases* (pp. 533-542).

[11] Chen, Y., & Revesz, P. (2003). Query spatiotemporal XML using DataFox. *Proceedings of ACM Internal Conference on Web Intelligence* (pp. 301-309).

[12] Bai, L. Y., Yan, L., & Ma, Z. M. (2014). Querying fuzzy spatiotemporal data using XQuery. *Integrated Computer-Aided Engineering*, *21(2)*, 147-162.

[13] Bai, L. Y., Yan, L., & Ma, Z. M. (2013). Determining topological relationship of fuzzy spatiotemporal data integrated with XML twig pattern. *Applied Intelligence*, *39(1)*, 75-100.

Luyi Bai received his PhD degree from Northeastern University, China. He is currently an associate professor at Northeastern University, 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.