Online Mining of Frequent Query Trees over XML Data Streams

Hua-Fu Li
Department of Computer Science,
National Chiao-Tung University
Hsinchu, Taiwan 300, R.O.C.
hfli@csie.nctu.edu.tw

Man-Kwan Shan
Department of Computer Science,
National Chengchi University
Taipei, Taiwan 116, R.O.C.
mkshan@cs.nccu.edu.tw

Suh-Yin Lee
Department of Computer Science,
National Chiao-Tung University
Hsinchu, Taiwan 300, R.O.C.
sylee@csie.nctu.edu.tw

ABSTRACT
In this paper, we proposed an online algorithm, called FQT-Stream (Frequent Query Trees of Streams), to mine the set of all frequent tree patterns over a continuous XML data stream. A new numbering method is proposed to represent the tree structure of a XML query tree. An effective sub-tree numeration approach is developed to extract the essential information from the XML data stream. The extracted information is stored in an effective summary data structure. Frequent query trees are mined from the current summary data structure by a depth-first-search manner.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – *data mining.*

General Terms: Algorithms.

Keywords: Web mining, data streams, XML, frequent query trees, online mining.

1. INTRODUCTION
In recent years, many applications generate data streams in real time, such as sensor data streams generated from sensor networks, online transaction flows in retail chains, Web record and click streams in Web applications, performance measurement in network monitoring, and traffic management, and call records in telecommunications.

A *data stream* is massive unbounded sequence of data elements continuously generated at a rapid rate. Mining of such data streams differs from the mining of traditional datasets in the following aspects [2]: First, each data element in the stream should be examined at most once, i.e., the proposed algorithm must be a *single-pass* algorithm. Second, the memory requirement of the proposed algorithm should be bounded even though new data elements are continuously generated from the data streams. Third, each data element in the stream should be processed as fast as possible. Fourth, the mined results should be instantly available when the user requested. Finally, the output errors should be constricted to be as small as possible.

Recently, several techniques have been developed to mine the set of all frequent itemsets [3, 4, 5, 6, 7, 8, 10] in data streams. However, less work on the field of mining complex data streams, such as XML data streams. In such a XML data stream, each incoming data element is a (XML) query tree structure. Asai et al. [1] proposed an online algorithm *StreamT* to analyze the frequent ordered trees from a continuous semi-structured data stream. Yang et al. [9] proposed online algorithms *XQPMiner* and *XQPMinerTID* to find the set of all frequent rooted ordered trees over a continuous XQuery stream.

In this paper, we proposed a novel one-pass algorithm, called *FQT-Stream* (Frequent Query Trees of Streams), to mine the set of all frequent query trees over a continuous XML data stream. An effective query sub-tree enumeration method is developed to extract the essential information from the stream. The extracted information is stored in an effective summary data structure, called *FQT-forest* (a forest of Frequent Query Trees). Frequent query trees are mined efficiently from the current FQT-forest by a depth-first-search manner.

The remainder of the paper is organized as follows. The problem definition is described in Section 2. In Section 3, we describe the design of our algorithm FQT-Stream. Finally, we conclude the work in Section 4.

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

2. PROBLEM DEFINITION
A (XML) **query tree stream** $QTS = QT_1, QT_2, \ldots, QT_N$ is a continuous sequence of query trees (QT), where N is the query identifier of *latest* query tree generated so far. The **support** of a query tree QT, denoted as **sup**(QT), is the number of query trees in S containing QT as a *sub-tree*. A query tree QT is called a **frequent query tree** (FQT) if and only if **sup**(QT) $\geq s \cdot N$, where s is a user-defined minimum support threshold in the range of [0, 1].

Problem Statement Given a continuous query tree stream QTS, and a minimum support threshold s in the range of [0, 1], the problem of frequent tree pattern mining over an online XML stream is to mine the set of frequent query trees by *one scan* of the query tree stream QTS.

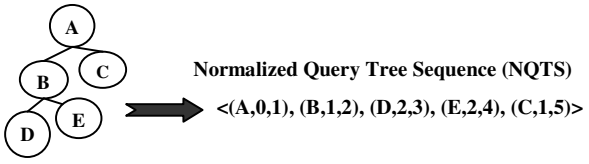

Figure 1. Transform a query tree (QT) into a normalized query tree sequence (NQTS).

3. THE PROPOSED ALGORITHM
The proposed algorithm FQT-Stream consists of five phases: read a query tree from the buffer in the main memory (phase 1), transform the query tree into a new Normalized Query Tree Sequence (NQTS) representation (phase 2), construct an in-memory summary data structure FQT-forest by projecting the NQTSs (phase 3), prune the infrequent information from the FQT-forest (phase 4), and find the set of all frequent query trees from the FQT-forest (phase 5). Since the phase 1 is straightforward, we shall focus on phases 2-5.

3.1 NQTS Transformation
For each incoming query tree QT, we transform it into a normalized query tree sequence ($NQTS$) using a depth-first-search (DFS) manner

on the *QT*. For example, a *QT* in Figure 1 is transformed into a *NQTS*: <(A,0,1), (B,1,2), (D,2,3), (E,2,4), (C,1,5)>, where the first field is the *node-id* in the QT, the second field is the *level* of the QT, and the third field is the *(sequence) order* of the NQTS. Note that the example NQTS is also called a *5-NQTS*, i.e., a NQTS with five nodes.

3.2 FQT-forest Construction

For each NQTS, two steps are performed to construct the summary data structure called FQT-forest (a forest of Frequent Query Trees).

Step 1: The NQTS is enumerated into several sub-sequences using *order-break* (OB) technique which is a level-wise method. We use an example NQTS, <(A,0,1), (B,1,2), (D,2,3), (E,2,4), (C,1,5)>, to describe the order-break technique.

First, the sequence <(A,0,1), (B,1,2), (D,2,3), (E,2,4), (C,1,5)> is broken into three 4-NQTSs, i.e., <(A,0,1), (D,2,3), (E,2,4), (C,1,5)>, <(A,0,1), (B,1,2), (E,2,4), (C,1,5)>, and <(A,0,1), (B,1,2), (D,2,3), (C,1,5)>. These sequences are belonged to 1-OB (1 Order Break). The term *1-OB* means that the sequence has one order break in the sequence order. For example, <(A,0,1), (D,2,3), (E,2,4), (C,1,5)> has an order break between (A,0,**1**) and (D,2,**3**), and <(A,0,1), (B,1,2), (E,2,4), (C,1,5)> has an order break between (B,1,**2**) and (E,2,**4**).

Second, these 4-NQTSs are enumerated into four 3-NQTSs with *one* order-break, i.e., <(A,0,1), (D,2,3), (E,2,4), (C,1,5)>➔<(A,0,1), (E,2,4), (C,1,5)>, <(A,0,1), (B,1,2), (E,2,4), (C,1,5)>➔<(A,0,1), (B,1,2), (C,1,5)> and <(A,0,1), (E,2,4), (C,1,5)>, and <(A,0,1), (B,1,2), (D,2,3), (C,1,5)>➔<(A,0,1), (B,1,2), (C,1,5)>. Since <(A,0,1), (E,2,4), (C,1,5)> and <(A,0,1), (B,1,2), (C,1,5)> appeared twice, we delete these duplicates. Hence, only *two* 3-NQTSs, i.e., <(A,0,1), (E,2,4), (C,1,5)> and <(A,0,1), (B,1,2), (C,1,5)>, are maintained.

Finally, these 3-NQTSs, <(A,0,1), (E,2,4), (C,1,5)>, and <(A,0,1), (B,1,2), (C,1,5)>, are enumerated into one 2-NQTS, <(A,0,1), (C,1,5)>, after removing the duplicates. Hence, the set of 1-OB contains 8 NQTSs, i.e., <(A,0,1), (D,2,3), (E,2,4), (C,1,5)>, <(A,0,1), (B,1,2), (E,2,4), (C,1,5)>, <(A,0,1), (B,1,2), (D,2,3), (C,1,5)>, <(A,0,1), (E,2,4), (C,1,5)>, <(A,0,1), (B,1,2), (C,1,5)> and <(A,0,1), (C,1,5)>. Note that <(A,0,1), (B,1,2), (D,2,3), (E,2,4), (C,1,5)> is called a 0-OB.

The set of 2-OB is generated from the set of 1-OB. For example, the NQTS with two order breaks, <(A,0,1), (D,2,3), (C,1,5)>, is generated from <(A,0,1), (D,2,3), (E,2,4), (C,1,5)>. Hence, the set of 2-OB contains only one NQTS: <(A,0,1), (D,2,3), (C,1,5)>.

Property 1 *The maximum size of order break is k–3, i.e., (k–3)-OB, if the query tree has k nodes.*

Step 2: These OBs (0-OB, 1-OB and 2-OB) are projected and inserted into the FQT-forest using the incremental projection technique proposed by Li et al. [5, 6]. The incremental projection method is described briefly as follows. A NQTS, $<x_1x_2\ldots x_i>$, with i nodes is converted into i sub-NQTSs; that is, $<x_i>$, $<x_{i-1}x_i>$, …, $<x_2x_3\ldots x_i>$, and $<x_1x_2\ldots x_i>$. Note that we use one field *node-id* to represent the fields (*node-id*, *level*, *sequence order*) for simplification. These sub-NQTSs are called *node-suffix NQTSs*, since the first node of each NQTS is a node-suffix of original NQTS. The operation is called *Incremental Projection* (IP) in this paper, and is denoted as **IP**(*NQTS*) = $|\{x_i|NQTS, x_{i-1}|NQTS, \ldots, x_1|NQTS\}$, where $x_j|NQTS = <x_jx_{j+1}\ldots x_i>$, $\forall j = 1, 2, \ldots, i$, and $NQTS = <x_1x_2\ldots x_i>|$. For example, the 1-OB, <(A,0,1), (D,2,3), (E,2,4), (C,1,5)>, is projected into four node-suffix NQTSs: <(C,1,5)>, <(E,2,4), (C,1,5)>, <(D,2,3), (E,2,4), (C,1,5)>, and <(A,0,1), (D,2,3), (E,2,4), (C,1,5)>. After projecting these OBs, a *tree structure checking* is performed as follows. *If the level of the first node in a node-suffix NQTS is not the smallest level, the node-suffix NQTS is deleted.* After the tree structure checking, these node-suffix NQTSs are inserted into a summary data structure FQT-forest, and the supports of corresponding nodes of FQT-forest are updated. FQT-forest consists of two parts: FN-list (a list of Frequent Nodes) and a set of NQTS-trees (a tree of Normalized Query Tree Sequences). Each unique node x in the query tree stream is stored in the FN-list and has a prefix tree with root-id x, denoted by x.NQTS-tree. In such a prefix tree-based summary data structure (NQTS-tree), a sequence is represented by a path and its appearance frequency, i.e., *support*, is maintained in the last node of the path.

3.3 Infrequent Information Pruning

In order to guarantee the limited space requirement, the infrequent information of FQT-forest is pruned after processing each incoming query tree. The prune operation consists of two steps. First, we check the support of each node x in the FN-list of FQT-forest. If its support, **sup**(x), is less than the $s \cdot N$, the prefix tree with root x is deleted. Second, we traverse the other prefix trees y.NQTS-tree ($y \neq x$) to find the infrequent NQTSs with the prefix x, and then prune them.

3.4 Frequent Query Trees Mining

Assume that there are k frequent nodes, namely $x_1, x_2, \ldots, x_k$, in the current FN-list. Let the minimum support threshold be s in the range of [0, 1], and the current length of XML data stream be N. For each entry x_i, $\forall i = 1, 2, \ldots, k$, in the FN-list, FQT-Stream traverses the x_i.NQTS-tree to find the sequences with prefix x_i whose estimated support is greater than or equal to $s \cdot N$ in a depth-first-search manner. Then, FQT-Stream stores these frequent query trees in a temporal list, called FQT-list (a list of Frequent Query Trees). Finally, FQT-Stream outputs the set of frequent query trees stored in the FQT-list. The operation is called *FQT-mining* (Frequent Query Trees mining).

4. CONCLUSIONS

In this paper, we proposed an efficient one-pass algorithm FQT-Stream (Frequent Query Trees of Streams) to discover the set of frequent query trees over the entire history of online XML data streams.

ACKNOWLEDGMENTS

The research is supported by National Science Council of R.O.C. under grant no. NSC94-2213-E-009-012.

REFERENCES

[1] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, et al., Online algorithms for mining semi-structured data stream. In Proc. ICDM, 2002.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, Models and issues in data stream systems. In Proc. PODS, 2002, pp. 1–16.

[3] J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In Proc. ACM SIGKDD, 2003, pp. 487-492.

[4] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities. In Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT, H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), 2003.

[5] H.-F. Li, S.-Y. Lee, and M.-K. Shan, An efficient algorithm for mining frequent itemsets over the entire history of data streams. In Proc. IWKDDS, 2004

[6] H.-F. Li, S.-Y. Lee, and M.-K. Shan, Online mining (recently) maximal frequent itemsets over data streams. In Proc. RIDE, 2005.

[7] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In Proc. VLDB, 2002, pp. 346-357.

[8] W.G. Teng, M.-S. Chen, and P. S. Yu. A regression-based temporal pattern mining scheme for data streams. In Proc. VLDB, 2003, pp. 93-104.

[9] L.H. Yang, M.L. Lee, and W. Hsu, Finding hot query patterns over an XQuery stream. VLDB Journal Special Issue on Data Stream Processing, 2004.

[10] J.-X. Yu, Z. Chong, H. Lu, and A. Zhou. False Positive or False Negative: Mining frequent itemsets from high speed transactional data streams. In Proc. VLDB, 2004, pp. 204-215.