On Mining Webclick Streams for Path Traversal Patterns

Hua-Fu Li
Department of Computer Science and Information Engineering
National Chiao-Tung University
No. 1001 Da-Shieh Rd., Hsinchu 300, Taiwan, R.O.C.
hfli@csie.nctu.edu.tw

Suh-Yin Lee
Department of Computer Science and Information Engineering
National Chiao-Tung University
No. 1001 Da-Shieh Rd., Hsinchu 300, Taiwan, R.O.C.
sylee@csie.nctu.edu.tw

Man-Kwan Shan
Department of Computer Science
National Cheng-Chi University
No. 64, Sec. 2, Zhi-Nan Rd., Wenshan, Taipei 116, Taiwan, R.O.C.
mkshan@cs.nccu.edu.tw

ABSTRACT
Mining user access patterns from a continuous stream of Web-clicks presents new challenges over traditional Web usage mining in a large static Web-click database. Modeling user access patterns as maximal forward references, we present a single-pass algorithm *StreamPath* for online discovering frequent path traversal patterns from an extended prefix tree-based data structure which stores the compressed and essential information about user's moving histories in the stream. Theoretical analysis and performance evaluation show that the space requirement of *StreamPath* is limited to a logarithmic boundary, and the execution time, compared with previous multiple-pass algorithms [2], is fast.

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

General Terms
Algorithms, Performance.

Keywords
Web-click streams, data stream mining, path traversal patterns

1. INTRODUCTION
Recently, database and data mining communities have focus on a new data model, where data arrives in the form of *continuous streams* [1]. In the streaming data model, data does not take the form of *persisten*t relations, but arrives *sequentially* (implicitly by *arrival time* or explicitly by *timestamp*), and is processed by an *online* algorithm whose workspace is insufficient to store all the data, so the main challenges of mining such streaming data (usually called *stream data mining*) is constrained by limited resources of *memory, processing time*, and *the total number of streaming data scan*. Applications include financial tickers, Web-log and click streams in Web applications, data feeds from sensor network, call detail records in telecommunications, online transactions in retail chains, etc. In this paper, we consider one of most important applications of data stream mining, namely, *cost-efficient mining path traversal patterns over Web-click stream*.

The problem of mining path traversal patterns in a large static Web-click dataset was proposed by Chen *et al*. [2] for Web usage mining. In this paper, we extend the problem of path traversal pattern into a streaming problem. The problem can be modified as follows. Let S be a continuous stream of Web-clicks, where a Web-click Wc consists of an identifier *UserID* of the Web user and a Web-page reference r accessed by the user, i.e., Wc = (*UserID*, r). In such streaming environment, a segment of Web-click stream arrived at timestamp t_i can be divided into a set of Web-click sequences. For example, a fragment of stream S = [(100, A)(100, B)(200, A)(300, B)(200, B)(200, C)(300, C)(100, D)(200, A)(200, E)$]^{t_i}$, arrived at timestamp t_i, can be classified into three Web-click sequences: {100, (A)(B)(D)}, {200, (A)(B)(C)(A)(E)}, and {300, (B)(C)}, where 100, 200, and 300 are the identifiers of Web users, and A, B, C, D, and E are references accessed by these users. For convenience, in the sequel, we drop the identifier of user, and denote a sequence of references {(A)(B)(D)} as {ABD}. A Web-click sequence $Wcs = <r_1, r_2, \ldots, r_k>$ consists of a sequence of forward references and backward references accessed by a Web user. A backward reference means revisiting a previously visited page by the same user access. A maximal forward reference MFR is a forward reference path without any backward references. Hence, we can convert a Web-click sequence into several maximal forward references, i.e., $Wcs = MFR_1, MFR_2, \ldots, MFR_i$, where $i \geq 1$. For instance, a Web-click sequence {$ABCAE$} consists of two maximal forward references, namely, <ABC> and <AE>. Therefore, we can map the problem of finding frequent path traversal patterns into the one of finding frequent occurring consecutive sequences (called *reference sequences*) among all maximal forward references. The *support* of a reference sequence Rs, denoted by $Sup(Rs)$, is the number of maximal forward references containing Rs divided by the total maximal forward references in S at timestamp t_i. A reference sequence Rs is called a *frequent traversal pattern* if $Sup(Rs) \geq MinSup$, where $MinSup$ is a minimum support threshold specified by the user. Consequently, the problem can be defined as follows. *Given a minimum support threshold MinSup and a continuous stream of Web-clicks S, the problem of mining Web-click streams for path traversal patterns is to discover the set of all frequent traversal patterns with respect to the characteristics of Web-click streams.*

The objective of this paper is to mine the set of all frequent traversal patterns over a Web-click stream by *one-scan* the stream with *limited memory usage* and *fast response time*. Our algorithm *StreamPath* has all of these characteristics, while none of previously published methods can claim the same.

2. ONLINE ALGORITHM
2.1 Pattern-Growth Mining
The framework of *StreamPath* is derived from the well-known pattern-growth algorithm called *FP-growth* proposed by Han *et al*. [3] for mining static databases, which is a divide-and-conquer method, and it can be divided into three phases. First, FP-growth scans the database to find all frequent items, and constructs a *Header-Table* to record the summary information of these frequent items. Second, FP-growth makes the second database scan to construct a conditional frequent-pattern tree (*FP-tree* for short), which is an extended prefix-tree structure for compressing the size of the original database. Third, FP-growth uses a recursive search scheme to generate all frequent itemsets from the FP-tree. More details about the FP-growth method can be found in [3].

There are following problems in developing a pattern-growth based algorithm for mining Web-click streams:

1. It requires scanning the database twice, i.e., one for Header-Table construction, and another for FP-tree construction.

WWW 2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

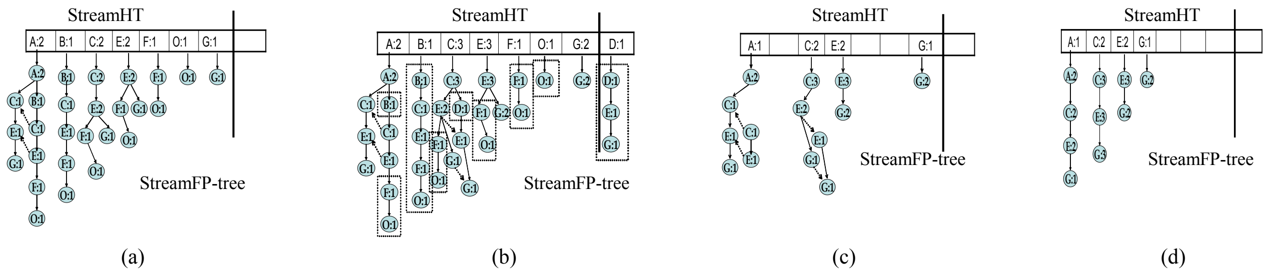

Figure 1. (a) StreamFP-tree after the first two maximal forward references, (b) StreamFP-tree reconstruction, (c) StreamFP-tree maintenance using StreamHT pruning, (d) Current status of StreamFP-tree after the first three maximal forward references {*ABCEFO*}, {*ACEG*}, and {*CDEG*}.

2. The upper bound of the FP-tree's memory usage is probably undetermined in such streaming environment.

These challenges show that static pattern-growth method can not appropriately meet the main performance requirements of mining Web-click streams. Hence, in this paper, we propose a modified pattern-growth framework, called *stream-efficient pattern-growth*, to satisfy the major performance requirements, namely, single-pass, bounded memory, and real-time, of data stream mining.

2.2 Stream-Efficient Pattern-Growth Mining

Algorithm *StreamPath* has two major features, namely *online dynamic maintaining* **StreamHT** (**Stream**ing **H**eader-**T**able), and *efficient constructing a* **StreamFP-tree** (**Stream**ing **F**requent-**P**attern **tree**) in a continuous stream of Web-clicks.

To illustrate these features of *StreamPath*, we will use the following example as a running example. Consider a fragment of example online Web-click stream *S* arrived at timestamp *t*; that is, *S*=[(100, *A*)(100, *B*)(200, *C*)(300, *C*)(100, *C*)(200, *D*)(100, *E*)(100, *F*)(200, *E*)(300, *D*)(100, *O*)(200, *D*)(100, *A*)(100, *C*)(300, *F*)(100, *E*)(200, *G*)(100, *G*)]^t. After a hashing function of each individual Web user, and the transform function of maximal forward references, we can obtain *three* Web-click sequences, namely, {*ABCEFOACEG*}, {*CDEG*}, and {*CDF*}, and *four* maximal forward references, i.e., <*ABCEFO*>, <*ACEG*>, <*CDEG*>, and <*CDF*>, where each capital letter indicates a Web-page reference, and we assume that the StreamHT contains at most seven frequent items, which is constrained by 1/*MinSup*. More details about maintaining frequent items over a streaming data can be found in [4]. After processing the first two maximal forward references in this stream, a StreamFP-tree and the StreamHT were constructed, as shown in Figure 1 (a). In Figure 1 (b), the StreamHT was broken (since the number of frequent items is greater than the maximal size of StreamHT, i.e., 8 > 7), while *StreamPath* reads in the third maximal forward reference {*CDEG*}. The nodes bounded by the dotted boxes were removed from the StreamFP-tree and reconstructs the StreamFP-tree and the StreamHT, as shown in Figure 1 (c) and Figure 1 (d), respectively. At this time, if a user query wants to output the current set of frequent traversal patterns, then *StreamPath* traverses the StreamFP-tree, as shown in Figure 1 (d), in depth first search (*DFS*) manner and generates a temporal list which containing the set of current (maximal) frequent traversal pattern *ACE* and *CEG*. From this running example, we can see that *StreamPath* has all of these characteristics, namely, single-scan, limited memory and real-time, in a streaming environment.

3. MEMORY ANALYSIS AND EXPERIMENTS

Let the StreamHT contains *k* items at any time. Therefore, we know there are at most $C^{k}_{\lfloor k/2\rfloor}$ frequent reference sequences in the current Web-click stream seen so far. If we construct a StreamFP-tree for all these frequent traversal patterns, the tree has height $\lfloor k/2\rfloor$. In the first level, there are $C^{\lfloor k/2\rfloor+1}_{1}$ nodes, in the second level, there are $C^{\lfloor k/2\rfloor+2}_{2}$ nodes, in the *i*-th level, there are $C^{\lfloor k/2\rfloor+i}_{i}$ nodes, and in the last level, the $\lfloor k/2\rfloor$ level, there are $C^{k}_{\lfloor k/2\rfloor}$ nodes. Thus, the total number of nodes is $C^{\lfloor k/2\rfloor+1}_{1} + C^{\lfloor k/2\rfloor+2}_{2} + C^{\lfloor k/2\rfloor+i}_{i} + \ldots + C^{k}_{\lfloor k/2\rfloor} = \sum_{i=1}^{\lfloor k/2\rfloor} C^{\lfloor k/2\rfloor+i}_{i}$ □

The space requirement of *StreamPath* consists of two parts: the working space needed to create a StreamHT, and the storage space needed for the StreamFP-tree construction. In worst case, the working space for StreamHT requires *k* entries. For storage, there are at most $\sum_{j=1}^{K}\sum_{i=1}^{\lfloor j/2\rfloor} C^{\lfloor j/2\rfloor+i}_{i}$ nodes of the StreamFP-tree. Thus, this gives a total space bound of $O(k+\sum_{j=1}^{K}\sum_{i=1}^{\lfloor j/2\rfloor} C^{\lfloor j/2\rfloor+i}_{i})$.

Dude to lack of space, we only summarize the comparisons of *StreamPath* and the algorithms, *FS* and *SS*, proposed in [2], using synthetic datasets same as [2]. On the synthetic datasets, *StreamPath* outperforms *FS* and *SS*. There are two main reasons why *StreamPath* does better than *FS* and *SS*. First, there is no candidate generation in *StreamPath*. Second, *StreamPath* needs only one streaming data scan.

4. CONCLUSIONS

The problem of mining data streams is much more complicated than traditional data mining due to the characteristics of streaming data. In this paper we present an efficient algorithm for mining path traversal patterns over Web-click streams. Based on our knowledge, algorithm *StreamPath* is a first efficient approach for Web-click stream mining satisfying the key properties of one streaming data scan, limited memory usage, and fast processing time for each streaming Web click.

ACKNOWLEDGEMENTS

The authors thank the reviewers for their valuable comments and suggestions with significantly improve the quality of this paper. The work was supported by the National Science Council of R.O.C. under grant no. NSC92-2213-E009-123.

REFERENCES

[1] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. Models and Issues in Data Stream Systems. In *Proc. of the 2002 ACM Symposium on Principles of Database Systems*, 2002.

[2] Chen, M.-S., Park, J.-S., and Yu, P. S. Efficient Data Mining for Path Traversal Patterns, *IEEE Transactions on Knowledge and Data Engineering* (*TKDE*), 10(2):209-221, 1998.

[3] Han, J., Pei, J., Yin, Y., and Mao, R. Mining Frequent Patterns without Candidate Generation: A Frequent-pattern Tree Approach. *Data Mining and Knowledge Discovery: An International Journal*, 8(1):53-87, 2004.

[4] Karp, R., Shenker, S., and Papadimitriou, C. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Transactions on Database Systems* (*TODS*), 28(1):51-55, 2003.