DSM-TKP: Mining Top-K Path Traversal Patterns over Web Click-Streams

Hua-Fu Li[a], Suh-Yin Lee[a], and Man-Kwan Shan[b]

[a]*Department of Computer Science and Information Engineering*
National Chiao-Tung University, Hsinchu 300, Taiwan
{hfli, sylee}@csie.nctu.edu.tw
[b]*Department of Computer Science*
National Chengchi University, Taipei 116, Taiwan
mkshan@cs.nccu.edu.tw

Abstract

Online, single-pass mining Web click streams poses some interesting computational issues, such as unbounded length of streaming data, possibly very fast arrival rate, and just one scan over previously arrived click-sequences. In this paper, we propose a new, single-pass algorithm, called DSM-TKP (Data Stream Mining for Top-K Path traversal patterns), for mining top-k path traversal patterns, where k is the desired number of path traversal patterns to be mined. An effective summary data structure called TKP-forest (Top-K Path forest) is used to maintain the essential information about the top-k path traversal patterns of the click-stream so far. Experimental studies show that DSM-TKP algorithm uses stable memory usage and makes only one pass over the streaming data.

1. Introduction

Recently, database and data mining communities have focused on a new data model, where data arrive in the form of *continuous streams*. It is often referred to as *data streams* or *streaming data*. Mining such streaming data poses some interesting computational issues, such as unknown or unbounded length of the stream, possibly very fast arrival rate, and inability to backtrack over previously arrived data elements [2, 7]. Many applications generate data streams in real time, such as sensor data generated from sensor networks, transaction flows in retail chains, Web record and click-streams in Web applications, performance measurement in network monitoring and traffic management, call records in telecommunications, and so on.

Mining clusters in evolving Web click-streams have been discussed in recent years [10, 11]. In this paper, we study the problem of mining top-k path traversal patterns in Web click-streams. The original problem of mining path traversal patterns from a large static Web click-dataset was proposed by Chen et al. [3]. Recently, Li et al. [6] proposed a first single-pass algorithm, called *StreamPath*, to mine the set of all path traversal patterns over continuous Web click-streams. In the framework of SteamPath algorithm, it requires a user-specified minimum support threshold *minsup*, and then mines path traversal patterns with estimated support values that are higher than the minimum support threshold. Unfortunately, the setting of minimum support threshold is quite tricky and it leads to the following problem that may hinder its popular use.

If the value of minimum support threshold is too small, the pattern mining algorithm may lead to the generation of thousands of patterns, whereas a too big one may often generate a few patterns or even no answers. As it is difficult to predict how many patterns will be mined with a user-defined minimum support threshold, the top-*k* pattern mining has been proposed.

The first top-*k* pattern mining algorithm Itemset-Loop was proposed by Fu et al. [5]. Itemset-Loop algorithm mines the *k* most frequent itemsets with lengths shorter than a user-defined value of *m*. LOOPBACK and BOMO are FP-tree-based top-*k* pattern mining algorithms [4], and uses the same estimated mechanism of Itemset-Loop. Moreover, experiments in [4] show that LOOPBACK and BOMO outperform the Itemset-Loop. TFP algorithm [11] is a FP-tree-based algorithm and mines the top-*k* closed frequent itemsets with lengths longer than a user-specified value of *min_l*. TSP [10] is the first algorithm to mine the top-k closed sequential patterns of lengths no less than the user-defined minimum length of mined patterns *min_l*.

Recently, Metwally et al. [9] proposed a single-pass algorithm to mine the top-*k* elements over data streams. However, the top-*k* elements are top-*k* items. In this paper, we propose an efficient single-pass algorithm called DSM-TKP (Data Stream Mining for Top-K Path traversal patterns) to mine the top-*k* path traversal patterns over Web click streams. An effective summary data structure called TKP-forest (Top-K Path forest) and an efficient structure pruning mechanism called KP (K Pruning) are proposed to overcome the data stream mining algorithm issues such as bounded space requirement and approximation. Based on our knowledge, DSM-TKP is the first single-pass algorithm for mining top-*k* path traversal pattern in streaming click-data.

2. Problem Statement

Let *S* be a continuous steam of Web clicks, where a Web click *wc* consists of Web user identifier (*Uid*) and a Web page reference *r* accessed by the user, i.e., *wc* = (*Uid*, *r*). In a steaming environment, a segment of Web click stream arrived at timestamp t_i can be divided into a set of Web click-sequences (or *click-sequences* in short). For example, a fragment of stream, *S* = [t_i, (100, *a*), (100, *b*), (200, *a*), (100, *c*), (200, *b*), (200, *c*), (100, *d*), (100, *e*), (200, *a*), (200, *e*)], arrived at timestamp t_i, can be divided into two click-sequences: <100, *abcde*>, and <200, *abcae*>, where 100, 200 are identifiers of Web users, and *a*, *b*, *c*, *d*, *e* are references accessed by these users. A **(Web) click-sequence** *CS* consists of a sequence of forward references and backward references accessed by a Web user. A **backward reference** means revisiting a previously visited reference by the same user. A **maximal forward reference** (MFR) is a forward reference path without any backward references. Hence, a click-sequence can be divided into several maximal forward references, i.e., *CS* = MFR_1, MFR_2, …, MFR_i, where $i \geq 1$. For example, a click-sequence <*abcae*> can be divided into two MFRs: <*abc*> and <*ae*>. Therefore, we can map the problem of mining top-*k* path traversal patterns into the one of finding top-*k* occurring consecutive sequences, called **reference sequences** (RSs), among all maximal forward references. The **support** of a reference sequence *RS*, denoted as *sup(RS)*, is the number of maximal forward references in the stream containing *RS* as a substring. A reference sequence is called **maximal** if it is not a substring of any other reference sequences. A maximal reference sequence is also called a **path traversal pattern**. A reference sequence *RS* is a **top-k maximal reference sequence** if there exists no more than (*k*-1) maximal reference sequences whose support is higher than that of *RS*. In this paper, our task is to mine top-*k* maximal reference sequences by one scan of a continuous stream of Web clicks when the value of *k* is given.

3. The Proposed Algorithm: DSM-TKP

The proposed algorithm DSM-TKP is composed of four steps: read a maximal forward reference from the buffer in the main memory (step 1), construct an in-memory summary data structure (step 2), prune and maintain the summary data structure (step 3), and find the path traversal patterns from the summary data structure so far (step 4). Steps 1 and 2 are performed in sequence for a new maximal forward reference. Steps 3 and 4 are usually performed periodically or when it is needed. Since the step 1 is straightforward, we shall henceforth focus on steps 2, 3, and 4, and devise algorithms for effective construction and maintenance of summary data structure, and efficient determination of path traversal patterns.

3.1 Effective Construction of the Summary Data Structure

In this section, we describe an algorithm which constructs the in-memory summary data structure called *Top-K Path forest.*

Definition 1 A *Top-K Path forest* (abbreviated as **TKP-forest**) is a prefix tree-based summary data structure defined below.

1. *TKP-forest* consists of a *K-References list* (abbreviated as **KR-list**), such as <r_1 r_2 … r_k>, and a set of *Local Path traversal pattern trees* (abbreviated as **LP-trees**) of references, denoted by r_i.LP-tree, $\forall i$ =1, 2, …, *k*, where r_i is the root node of r_i.LP-tree.
2. Each node in the r_i.*LP-tree*, $\forall i$ =1, 2, …, *k*, consists of four fields: *fid*, *esup*, *mfr_id*, and *node-link*, where *fid* is the identifier of the incoming maximal forward reference, *esup* registers the number of maximal forward references represented by a potion of the path reaching the node with the *fid*, the value of *mfr_id* assigned to a new node is the identifier of current maximal forward reference, and *node-link* links up a node with the next node with the same *fid* in the same LP-tree or null if there is none.
3. Each entry in the *KR-list* consists of four fields: *fid*, *esup*, *mfr_id*, and *head-link*, where *fid* registers which reference identifier the entry represents, *esup* records the number of maximal forward references containing the reference carrying the reference id, the *mfr_id* assigned to a new entry is the identifier of current maximal forward reference, and *head-link* is a pointer, and points to the root node of the *fid*.LP-tree.

The construction algorithm of TKP-forest is shown in Figure 1. The scenario of TKP-forest construction is described as follows. First of all, DSM-TKP reads a maximal forward reference *MFR* = <r_1r_2 …r_m>, for example, from the buffer in the main memory, projects the MFR into *m* sub-maximal forward references (abbreviated as **sub-MFRs**), and inserts these sub-MFRs into the TKP-forest as branches. Note that *m* is the number of references in the maximal forward reference. The projection of each incoming maximal forward reference is described as follows. Each maximal forward reference, *MFR* = <r_1 r_2 … r_m>, is converted into *m* sub-MFRs; that is, < r_1 r_2 … r_m >, < r_2r_3 … r_m >, …, and < r_m >. These *m* sequences are called *reference-suffix maximal forward references* (abbreviated as **rs-MFRs**), since the first reference of each sequence is a suffix of the original maximal forward reference. The projection step is called *maximal forward reference projection*, and denoted by **MFR-projection** (*MFR*) = {r_1|MFR, r_2|MFR, …, r_i|MFR, … , r_m|MFR}, where r_i|MFR = <$r_i r_{i+1} \ldots r_m$>, $\forall i$ = 1, 2, …, *m*. The cost of this projection is $(m^2+m)/2$, i.e., $m + (m-1) + \ldots + 2 + 1$.

After performing the MFR-projection, DSM-TKP algorithm inserts the *MFR* into the KR-list, and then removes it from the buffer in the main memory. Next, the set of rs-MFRs are inserted into the r_i.LP-trees ($\forall i$ =1, 2, …, *m*) as branches. If a MFR shares a prefix with a MFR already in the LP-tree, the new MFR will share a prefix of the branch representing that MFR. Moreover, an estimated support counter is associated with each node in the tree. The counter is updated when a rs-MFR causes the insertion of a new branch. The step is called the *rs-MFR insertion*.

Algorithm 1 (TKP-forest construction)
Input: A stream of maximal forward references, S = [MFR_1, MFR_2, …, MFR_N), a user-specified value *k*.
Output: A TKP-forest generated so far.

```
1:  KR-list = { };  /*initialize the KR-list to empty.*/
2:  foreach $MFR_i$, = <$x_1 x_2 \cdots x_m$>, do
                        /* $m \geq 1$, $i$=1, 2, …, $N$ */
3:     foreach reference $x_j \in MFR_i$ do
4:        if $x_j \notin$ KR-list then
5:           create a new entry of form ($x_j$, 1, $i$, head-link)
             into the KR-list;
6:        else /* the entry already exists in the KR-list*/
7:              $x_j.esup = x_j.esup + 1$;
8:        end if
9:     end for
10:    call MFR-Projection($MFR_i$);
11:    call rs-MFR insertion;
12: end for
13: call TKP-forest-pruning(TKP-forest, k);
        /* Step 3 of DSM-TKP algorithm */
14: end for
```

Figure 1: Algorithm of TKP-forest Construction

Example 1. Let the first six maximal forward references be < *abcde* >, < *acd* >, < *cef* >, < *acdf* >, < *cef* >, and < *df* >, where *a*, *b*, *c*, *d*, *e* and *f* are references in the stream. The TKP-forest with respect to the first two MFRs, < *abcde* > and < *acd* >, constructed by DSM-TKP algorithm is shown in Figure 2 and Figure 3, respectively.

3.2 Effective Pruning of the Summary Data Structure

The TKP-forest pruning mechanism used in DSM-TKP is performed when the number of references in the KR-list is greater than *k*. The pruning mechanism used in DSM-TKP algorithm is extended from the work [8], and shown in Figure 4.

The next step of DSM-TKP is to determine the top-*k* path traversal patterns from the current TKP-forest. The step is performed only when the analytical results of the stream is requested.

3.3 Determination of the Top-*k* Path Traversal Patterns

Assume that there are *k* references, namely r_1, r_2, …, r_k, in the current KR-list. For each entry r_i, $\forall i$ =1, 2, …, *k*, in the KR-list, DSM-TKP algorithm traverses the r_i.LP-tree to find the estimated support of each reference sequence with a prefix r_i in a depth-first-search (DFS) manner. Then, DSM-TKP stores these reference sequences into a temporal list of candidate maximal reference sequences, i.e., path traversal patterns, in a support decreasing order. Finally, DSM-TKP outputs the first *k* maximal reference sequences from the temporal list. For example, in Figure 5, the top-3 path traversal patterns are <*acd*: 3>, <*cef*: 2>, and <*df*: 2>, where the 3-th largest estimated support in the reordered KR-list is 2.

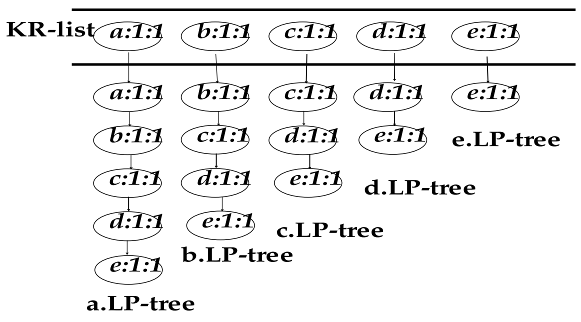

Figure 2: TKP-forest construction after processing the first maximal forward reference < *abcde* >

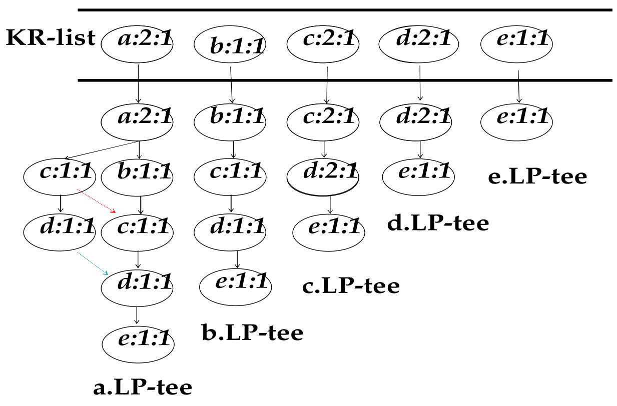

Figure 3: TKP-forest construction after processing the second maximal forward reference < *acd* >

```
Subroutine TKP-forest-pruning(TKP-forest, k)
1: sort the references, $r_1$, $r_2$, …, $r_{k'}$, in the KR-list and
   reorder the references in an estimated support
   decreasing order, i.e., $r_1$', $r_2$', …, $r_{k'}$', where $sup(r_1') \geq$
   $sup(r_2') \geq \cdots \geq sup(r_k')$;
2: find $r_{KL}$' in the reordered KR-list;
    /* $r_{KL}$' be a reference whose estimated support is the
       k-th largest one in the KR-list; */
3: foreach $r_i$'∈ KR-list, $\forall i = 1, 2, \ldots, KL$ do
4:         $esup(r_i') = esup(r_i') - esup(r_{KL-1}')$;
5: endfor
6: foreach $r_j$'∈ KR-list, $\forall j = KL+1, KL+2, \ldots, k'$ do
7:         delete $r_j$' from the current KR-list;
8:         delete $r_j$'.LP-tree;
9: endfor
```

Figure 4: Algorithm of TKP-forest pruning

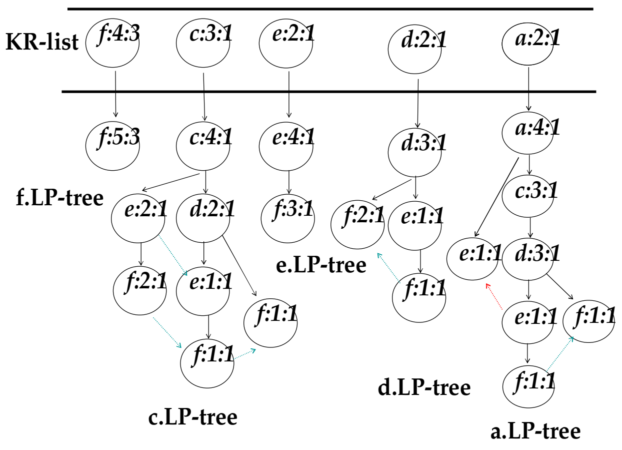

Figure 5: TKP-forest example

4. Conclusions

In this paper, we propose an efficient, online algorithm, DSM-TKP (Data Stream Mining for Top-K Path traversal patterns), for mining top-k maximal reference sequences in an infinite sequence of Web click-sequences. An effective summary data structure, TKP-forest (Top-K Path forest), is developed to store the essential information about the set of top-k path traversal patterns of the stream so far. An efficient pruning mechanism of TKP-forest is used to guarantee that the upper bound of the summary data structure is predictable. Based on best knowledge, DSM-TKP algorithm is the first single-pass algorithm for mining top-k path traversal patterns.

Acknowledgement

The authors thank the reviewers' precious comments for improving the quality of the paper. The research is supported by National Science Council of R.O.C. under grant no. NSC93-2213-E-009-043.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In Proc. of the 20th International Conference on Very Large Data Based (VLDB), 1994, pp. 487-499.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *in Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (*PODS'02*), 2002.

[3] M.-S. Chen, J.-S. Park and P. S. Yu, "Efficient Data Mining for Path Traversal Patterns," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 10, No. 2, April, 1998, pp. 209-221.

[4] Y.L. Cheung, A. W.-C. Fu, "Mining Association Rules without Support Threshold: with and without Item Constraints", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 9, September, 2004, pp 1052-1069.

[5] A. W.-C. Fu, R. W.-W. Kwong, and J. Tang, "Mining N-most Interesting Itemsets," in *Proc. of 12th International Symposium on Methodologies for Intelligent Systems* (*ISMIS'00*), October 11-14, 2000.

[6] H.-F. Li, S.-Y. Lee and M.-K. Shan, "On Mining Webclick Streams for Path Traversal Patterns", *in Proc. of the 13th World Wide Web Conference* (*WWW'04*), New York, 2004.

[7] L. Golab and M. T. Ozsu, "Issues in Data Stream Management," *In SIGMOD Record*, Vol. 32, No. 2, June 2003, pp. 5-14.

[8] R. M. Karp, S. Shenker, and C. H. Paradimitriou, "A Simple Algorithm for Finding Frequent Elements in Streams and Bags," *ACM Transactions on Database Systems*, Vol. 28, No. 1, March 2003, pp. 51-55.

[9] A. Metwally, D. Agrawal, A. E. Abbadi, "Efficient Computation of Frequent and Top-k Elements in Data Streams," in *Proc. of 10th International Conference on Database Theory* (*ICDT*'05), pp. 398-412, 2005.

[10] O. Nasraoui, C. Cardona, C. Rojas, and F. Gonzalez, "Mining Evolving User Profiles in Noisy Web Clickstream Data with a Scalable Immune System Clustering Algorithm", in Proc. of WebKDD 2003 – KDD Workshop on Web mining as a Premise to Effective and Intelligent Web Applications, pp. 71-81, 2003.

[11] O. Nasraoui, C. Cardona, C. Rojas, F. González, "TECNO-STREAMS: Tracking Evolving Clusters in Noisy Data Streams with a Scalable Immune System Learning Model", in Proc. of 3rd *IEEE International Conference on Data Mining* (*ICDM'03*), pp. 235-242, 2003.

[12] P. Tzvetkov, X. Yan, J. Han, "TSP: Mining Top-K Closed Sequential Patterns," in *Proc. of the 3rd IEEE International Conference on Data Mining* (*ICDM'03*), pp.347-354, 2003.

[13] J. Wang, J. Han, Y. Lu, and P. Tzvetkov, "TFP: An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 17, No. 5, May 2005, pp. 652-664.