Mining Top-K Path Traversal Patterns over Streaming Web Click-Sequences*

HUA-FU LI[1,2] AND SUH-YIN LEE[2]
[1]*Department of Computer Science*
Kainan University
Taoyuan, 338 Taiwan
E-mail: hfli@mail.knu.edu.tw
[2]*Department of Computer Science*
National Chiao Tung University
Hsinchu, 300 Taiwan
E-mail: {hfli; sylee}@csie.nctu.edu.tw

Online, one-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 Web 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 a set of 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 (a forest of Top-K Path traversal patterns), is used to maintain the essential information about the top-*k* path traversal patterns generated so far. Experimental studies show that the proposed DSM-TKP algorithm uses stable memory usage and makes only one pass over the streaming Web click-sequences.

Keywords: web usage mining, data streams, path traversal patterns, top-*k* pattern mining, single-pass mining

1. INTRODUCTION

In recent years, 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 streams, 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.

Recently, online mining of clusters in evolving Web click-streams have been discussed [10, 11]. In this paper, we study the research issue of mining top-*k* path traversal patterns over 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]. Li *et al.* [6] proposed a first single-pass algorithm **DSM-PLW** to mine the set of all path traversal pat-

Received August 23, 2007; revised April 21, 2008; accepted June 26, 2008.
Communicated by Makoto Takizawa.
* A preliminary version in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence, France, September 19-22, 2005. The research was supported in part by the National Science Council of Taiwan, R.O.C., Project No. 95-2221-E-009-069-MY3 and NSC 96-2218-E-424-001.

terns over continuous Web click-streams. In the framework of DSM-PLW algorithm, it requires a user-specified minimum support threshold *minsup*, and mines the 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 frequent 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]. Based on Apriori property [1], 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 use the same estimated mechanism of Itemset-Loop. Moreover, experiments in [4] show that LOOPBACK and BOMO outperform the Itemset-Loop. **TFP** algorithm [13] 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** [12] 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*. All above algorithms are multiple-pass data mining approaches. Therefore, these methods can not be used to mine patterns from streaming data.

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 a set of top-*k* path traversal patterns over Web click streams. An effective summary data structure, called **TKP-forest** (a forest of Top-K Path traversal patterns), and an efficient memory pruning mechanism, called **TKP-forest-Maintenance** (Maintenance of TKP-forest), are proposed to overcome the issues of mining data streams 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 over streaming Web click-sequences.

The remainder of this paper is organized as follows. The problem of mining top-*k* path traversal patterns over streaming click-sequences is defined in section 2. In section 3, the DSM-TKP algorithm is proposed to mine a set of top-*k* path traversal patterns. Comprehensive experiments of the proposed algorithm are discussed in section 4. Finally, we conclude the work in section 5.

2. PROBLEM DEFINITION

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*). 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, *i.e.* Web pages, accessed by these users.

A **click-sequence** (*CS*) consists of a sequence of forward references and backward references accessed by a Web user. A **backward reference** (*BR*) 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 *CS* can be divided into several maximal forward references, *i.e.*, $CS = \{MFR_1, MFR_2, \ldots, MFR_i\}$, where $i \geq 1$. For example, a click-sequence <*abcae*> can be divided into two MFRs, *i.e.*, <*abc*> and <*ae*>. The number of references of a MFR is called the **size** of MFR. For example, the size of MFR <*abc*> is 3. A **MFR stream** (*MS*) is a stream of maximal forward references, *i.e.*, $MS = [MFR_1, MFR_2, \ldots, MFR_N)$, where *N* is the identifier of latest incoming maximal forward reference MFR_N.

Therefore, we can map the problem of mining top-*k* path traversal patterns into the one of finding top-*k* occurring consecutive sequences, called **top-*k* reference sequences**, among a MFR stream.

The **estimated support** of a reference sequence (*RS*), denoted as *esup*(*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[1] no more than $(k - 1)$ maximal reference sequences whose support is higher than that of *RS* and size is grater than one.

In this paper, our task is to mine top-*k* maximal reference sequences by one scan of a Web click-sequence stream when the value of *k* is given.

3. DSM-TKP: DATA STREAMS MINING FOR TOP-K PATH TRAVERSAL PATTERNS

The proposed algorithm, called **DSM-TKP** (Data Stream Mining for Top-K Path traversal patterns), 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 top-*k* path traversal patterns from the current summary data structure (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 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, an efficient algorithm is proposed to construct the in-memory summary data structure, called **TKP-forest** (a forest of Top-K Path traversal patterns).

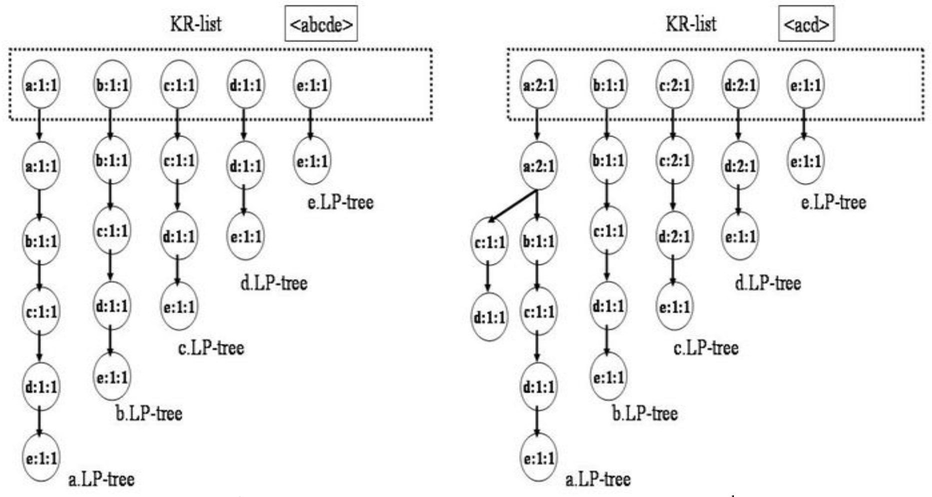
[1] Since there could be more than one pattern having the same support in a stream, to ensure the set mined is independent of the ordering of the references and MFRs, our method will mine every path traversal pattern whose support is no less than the *k*th path traversal patterns.

Definition 1 A TKP-forest is a prefix tree-based summary data structure defined below.

1. *TKP-forest* consists of a *list of K-References* (abbreviated as **KR-list**), such as $<r_1r_2 \ldots r_k>$, and a set of *Local Path-traversal-pattern trees* (abbreviated as **LP-trees**) of references of KR-list, denoted by r_i.LP-tree, $\forall i = 1, 2, \ldots, k$, where r_i is the root node of r_i.LP-tree.
2. Each node in the r_i.*LP-tree*, $\forall i = 1, 2, \ldots, k$, 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 a pointer *node-link* points to the next node 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 identifier, 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 given in Fig. 1. The scenario of TKP-forest construction is described as follows. First of all, the proposed DSM-TKP algorithm reads a maximal forward reference $MFR = <r_1r_2 \ldots r_m>$, for example, from the buffer in the main memory, projects it 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 *size* of the maximal forward reference.

```
Algorithm TKP-forest construction
Input: A stream of maximal forward references, $MS = [MFR_1, MFR_2, \ldots, MFR_N)$, a
user-specified value $k$.
Output: A TKP-forest generated so far.
1:  KR-list = {};   /* initialize the KR-list to empty. */
2:  foreach $MFR_i$, = $<x_1x_2 \ldots x_m>$, do   /* $m \geq 1, i = 1, 2, \ldots, N$ */
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(rs-MFRs of $MFR_i$);
12:   end for
13:   call TKP-forest-Maintenance(TKP-forest, $k$);   /* Step 3 of DSM-TKP algorithm */
14:   end for
```
Fig. 1. Construction algorithm of the proposed summary data structure TKP-forest.



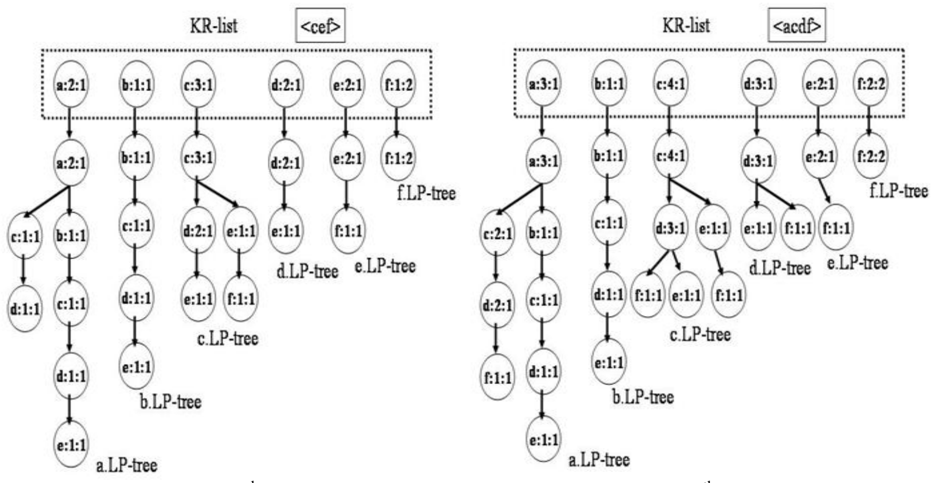
(a) Processing 1st MFR <*abcde*>. (b) Processing 2nd MFR <*acd*>.

Fig. 2. Construction of TKP-forest after processing the 1st MFR and 2nd MFR.

The details of projection of each incoming MFR are described as follows. First, each incoming maximal forward reference, *MFR* = <$r_1r_2 \dots r_m$>, is converted into *m* sub-MFRs; that is, <$r_1r_2 \dots r_m$>, <$r_2r_3 \dots r_m$>, …, 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 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_ir_{i+1}, …, r_m>, $\forall i = 1, 2, \dots, m$. Therefore, the cost of **MFR-projection**(*MFR*) is $(m^2 + m)/2$, *i.e.*, $m + (m - 1) + \dots + 2 + 1$, where the size of MFR is *m*.

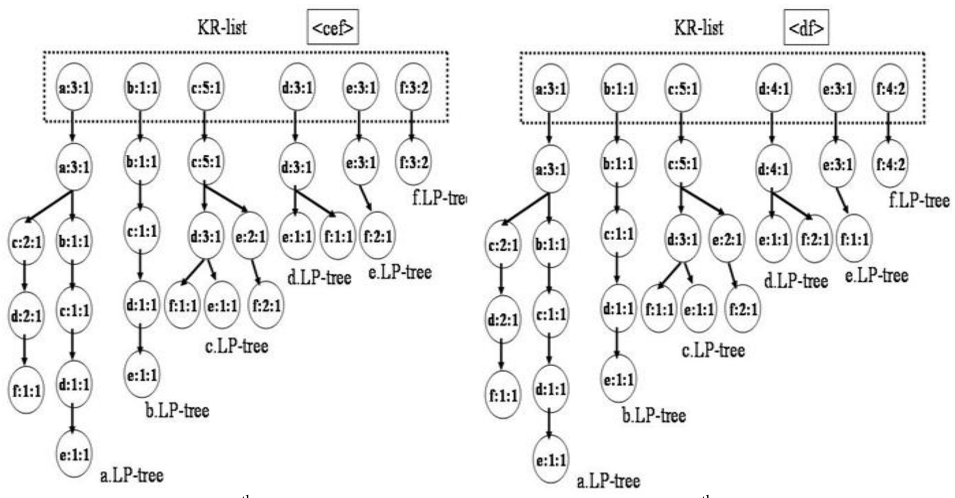
After performing MFR-projection, DSM-TKP algorithm inserts each reference of *MFR* into the KR-list, and removes it from the buffer in the main memory. Next, the set of rs-MFRs are inserted into the r_i.LP-trees ($\forall i = 1, 2, \dots, m$) as branches. If a MFR shares a prefix with a MFR already in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

Example 1: Let the first six maximal forward references of an example stream be <*abcde*>, <*acd*>, <*cef*>, <*acdf*>, <*cef*>, and <*df*>, where *a*, *b*, *c*, *d*, *e* and *f* are Web pages accessed by Web users. After performing **MFR-projection**(<*abcde*>) and **MFR-projection** (<*acd*>), five rs-MFRs, *i.e.*, <*abcde*>, <*bcde*>, <*cde*>, <*de*>, and <*e*>, of 1st MFR <*abcde*> and three rs-MFRs, *i.e.*, <*acd*>, <*cd*> and <*d*>, of 2nd MFR <*acd*> are inserted into the TKP-forest as shown in Figs. 2 (a) and (b), respectively. The results of TKP-forest constructed by DSM-TKP algorithm with respect to 3rd MFR <*cef*> and 4th MFR <*acdf*> are given in Figs. 3 (a) and (b), respectively. The final results of TKP-forest construction with respect to 5th MFR <*cef*> and 6th MFR <*df*> of Example 1 are given in Figs. 4 (a) and (b), respectively.



(a) Processing 3rd MFR <*cef*>. (b) Processing 4th MFR <*acdf*>.

Fig. 3. Construction of TKP-forest after processing the 3rd MFR and 4th MFR.



(a) Processing 5th MFR <*cef*>. (b) Processing 6th MFR <*df*>.

Fig. 4. Construction of TKP-forest after processing the 5th MFR and 6th MFR.

3.2 Effective Pruning of the Summary Data Structure

The effective pruning mechanism of TKP-forest, called **TKP-forest-Maintenance** (Maintenance of TKP-forest), used in the DSM-TKP algorithm is performed when the number of references in the KR-list is greater than *k*. The pruning mechanism TKP-forest-Maintenance is extended from the work [8], and is shown in Fig. 5. The pruning method is composed of four steps and is described as follows.

First, TKP-forest-Maintenance sorts and reorders the references, for example, r_1, r_2, …, r_k, of KR-list in an estimated support decreasing order, *i.e.*, $esup(r_{1'}) \geq esup(r_{2'}) \geq \ldots \geq esup(r_{k'})$. Second, the pruning method find the final *k*th largest reference, *i.e.*, $r_{KL'}$, from the reordered KR-list. Note that if there are more than one reference has the same *k*th largest estimated support, we take the right-most reference as $r_{KL'}$. Third, all information about the other references, $r_{j'}$, where $j = KL + 1, KL + 2, \ldots, k'$, are removed from the current TKP-forest, *i.e.*, remove $r_{j'}$ from the current KR-list and prune the r_j.LP-tree. Finally, the estimated support of each reference of the KR-list is decreased by the estimated support of the reference $r_{KL\text{-}1'}$. After performing these steps, the set of top-*k* path traversal patterns are maintained in the current TKP-forest.

Now, we use the TKP-forest of Fig. 4 (b) of Example 1 to demonstrate the pruning mechanism TKP-forest-Maintenance. The result of the first two steps of pruning mechanism is shown in Fig. 6. From this figure, we can find that the right-most reference (*e*) with *esup*(*e*) = 3 where value 3 is the third largest estimated support in the reordered KR-list. The result of steps 3 and 4 of TKP-forest-Maintenance is given in Fig. 7. From this figure, we can see that all information about reference (*b*), *i.e.*, reference (*b*) of KR-list and its *b*.LP-tree, are pruned from the current TKP-forest since reference (*b*) is not a candidate of top-*k* path traversal patterns at this moment.

The next step of DSM-TKP algorithm is to find the set of top-*k* path traversal patterns from the current TKP-forest. The step is performed only when the analytical results of the stream of Web click-sequences is requested.

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

Fig. 5. Pruning algorithm TKP-forest-maintenance of the proposed summary data structure TKP-forest.

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, $\forall i = 1, 2, \ldots, k$, in the KR-list, the 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

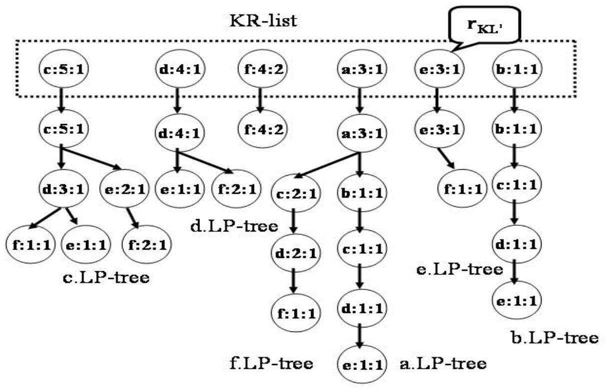


Fig. 6. TKP-forest of example 1 after steps 1 and 2 of TKP-forest-maintenance.

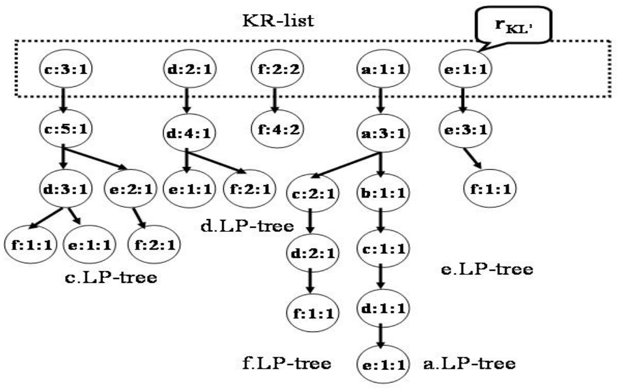


Fig. 7. TKP-forest of example 1 after steps 3 and 4 of TKP-forest-maintenance.

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. The temporal list is called **CTKP-list** (a list of Candidate Top-K Path traversal patterns). Finally, DSM-TKP outputs the first *k* maximal reference sequences from the CTKP-list. For example, the CTKP-list generated by DSM-TKP is composed of ten path traversal patterns, *i.e.*, <*cd*: 3>, <*cef*: 2>, <*df*: 2>, <*acd*: 2>, <*cdf*: 1>, <*cde*: 1>, <*de*: 1>, <*acdf*: 1>, <*abcde*: 1>, and <*ef*: 1>. Note that Since there could be more than one pattern having the same support in a stream, to ensure the set mined is independent of the ordering of the references and MFRs, our method will mine every path traversal pattern whose support is no less than the *k*th path traversal patterns. Consequently, the top-3 path traversal patterns are <*cd*: 3>, <*cef*: 2>, <*df*: 2>, and <*acd*: 2>.

3.4 Space Analysis of the Proposed Summary Data Structure TKP-forest

The space upper bound of DSM-TKP algorithm for constructing a prefix tree-based in-memory summary data structure is discussed in this section. Because the in-memory data structure TKP-forest used in DSM-TKP is based on the DSM-PLW algorithm [6], the space upper bound of DSM-TKP algorithm discussed as follows is the same as that of DSM-PLW algorithm in worst case.

Theorem 1 A prefix tree-based summary data structure has at most 2^k nodes for storing the set of top-k path traversal patterns of Web click streams.

Proof **[6]:** Let k be the number of frequent references in the current TKP-forest. According to the well-known Apriori property [1], the number of candidate frequent reference sequences is C(k, 1) regarding path traversal patterns with size 1, C(k, 2) regarding path traversal patterns with size 2, …, C(k, i) regarding path traversal patterns with size i, ..., and C(k, k) regarding path traversal patterns with size k references. In a prefix tree-based summary data structure, a reference sequence is represented by a path and its appearance support is maintained in the last node of the path. Hence, there are C(k, 1) nodes in the 1st level, C(k, 2) nodes in the 2nd level, …, C(k, i) nodes in the ith level, …, and C(k, k) nodes in the kth level. There are totally C(k, 1) + C(k, 2) + … + C(k, i) + … + C(k, k) nodes in the prefix tree-based summary data structure. Consequently, the space upper bound of any proposed prefix tree-based summary data structure for mining top-k path traversal patterns is $O(2^k)$ in worst case.

4. PERFORMANCE EVALUATION

4.1 Generation of Synthetic Data

All the experiments are performed on a 1.80 GHz Pentium 4 processor with 1 GB main memory, running on Microsoft Windows 2000. In addition, all the programs are written in Microsoft/Visual C++ 6.0. To evaluate the performance of the proposed DSM-TKP algorithm, two experiments are performed. The experiments were carried out on the synthetic data generator of Web click-sequences used in [10]. We describe it briefly as follows [6, 11]. A traversal tree is constructed to mimic Web site structure whose starting position is a root node of the tree. The traversal tree is composed of internal nodes and leaf nodes. A traversal path consists of nodes accessed by a Web user. The size of each traversal path is picked from a Poisson distribution with mean equal to $|P|$, where $|P|$ is the average size of reference paths. With the first node being the root node, a traversal path is generated probabilistically within the traversal tree as follows. Each edge connecting to an internal node is assigned with a weight. The weight corresponds to the probability that each edge will be next accessed by the Web user. The weight to its parent node is assigned with p_0, which is generally $1/(n + 1)$ where n is the number of child nodes. The probability of traveling to each child node, p_i, is determined from an exponential distribution with unit mean. Moreover, the probability is normalized that the sum of the weights for all child nodes is equal to $1 - p_0$. When the path arrives at a leaf node, the next move would be either to its parent node in backward (with a default probability 0.25) or to any internal node (with an aggregate probability 0.75). More detail about the generation of synthetic traversal paths can be found in [11].

Two synthetic data streams, denoted by $H10.I5.D100K$ and $H10.P15.D100K$ are studied in this section. The first one $H10.I5$ has a traversal tree with height of 10 levels and has average size of the reference paths with 5 references. The second one *H*10.*P*15 has average size of the reference paths with 15 references. Both datasets consist of 100,000 Web click-sequences (*D*100*K*). In all experiments, the click-sequences of each dataset are looked up in sequence to simulate the environment of a continuous data stream [6].

4.2 Experimental Results of Execution Time and Memory Usages of DSM-TKP

Performance evaluation of execution time and memory usage of the proposed DSM-TKP algorithm are discussed in this section. We evaluated the effect of various *k* values for both synthetic data streams, *H*10.*I*5 and *H*10.*I*15, with 100K click-sequences.

First, in the experiments of execution time, two parts are evaluated: (a) the construction time of TKP-forest for measure the performance of online processing, (b) the pruning time and the top-*k* pattern generation time for measure the performance of analysis. In Fig. 8, we plot construction time taken by our algorithm for values of *k* ranging from 1,000 to 200. The figure shows how decreasing *k* leads to increase in construction time of TKP-forest. Moreover, in Fig. 9, the execution time of TKP-forest pruning and top-*k* pattern generation of DSM-TKP algorithm is evaluated to measure the performance of analysis. From Figs. 8 and 9, we can find that the proposed DSM-TKP algorithm has good adaptability for online processing and streaming analysis under various *k* values.

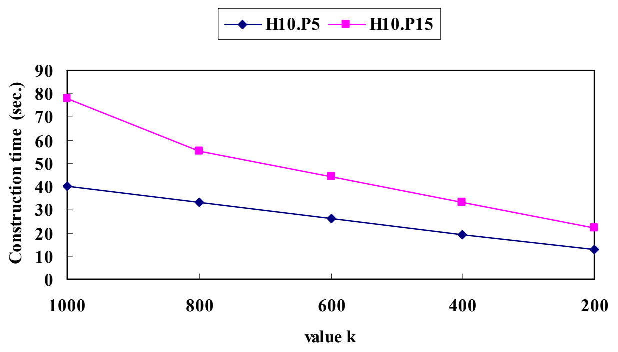


Fig. 8. Performance comparisons of construction time of TKP-forest over various *k* values, *D* = 100K.

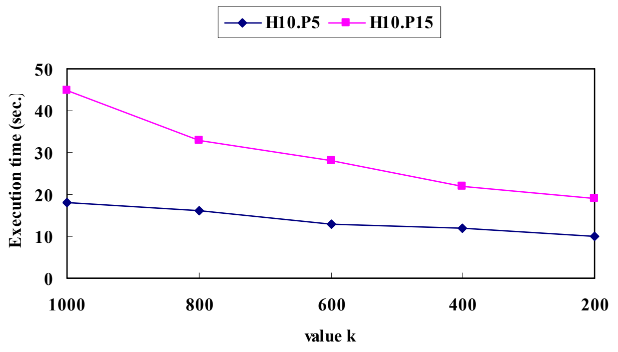


Fig. 9. Performance comparisons of pruning time and top-*k* pattern generation time over various *k* values, *D* = 100K.

To assess the scalability of our algorithm, scale-up experiments were conducted. Fig. 10 shows that the processing time of DSM-TKP algorithm increases linearly as the size of click-sequences increases, ranging from 100K to 1,000K, where *k* = 1,000. Processing time is composed of construction time, pruning time and top-*k* path traversal pattern generation time. Hence, from this figure, we can see that the proposed DSM-TKP algorithm exhibits good linearity in scale-up.

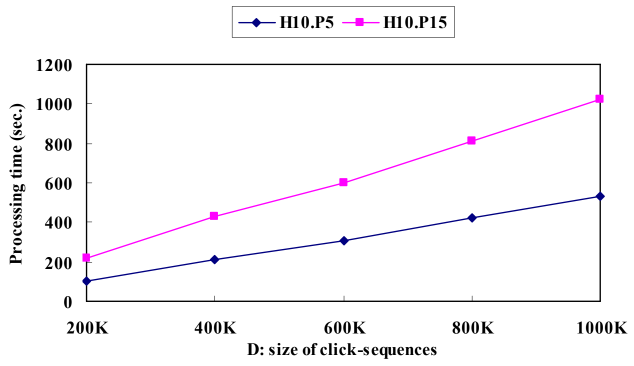


Fig. 10. Linear scalability of the sizes of click-sequence from 100K to 1,000K (*k* = 1,000).

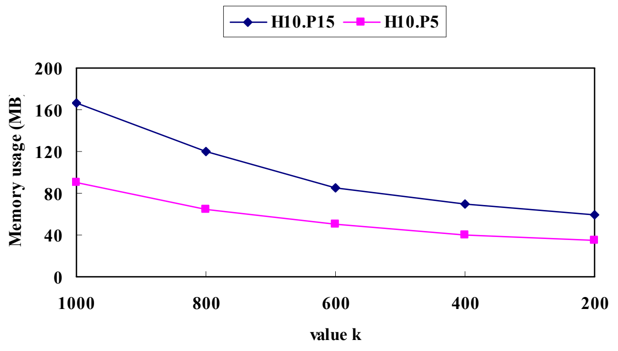


Fig. 11. Performance comparisons of memory usages over various *k* values, D = 200K.

The result of Fig. 11 is the memory requirement of the proposed algorithm for mining top-*k* path traversal patterns under various *k* values. From this figure, we can find that the memory usage of DSM-TKP algorithm is linearly affected by the sized of value *k*. The last experiment of DSM-TKP algorithm is the evaluation of precision. In Fig. 12, we can find that the proposed algorithm has good precision under various *k* values.

5. CONCLUSIONS

In this paper, we propose an efficient, online algorithm, called **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

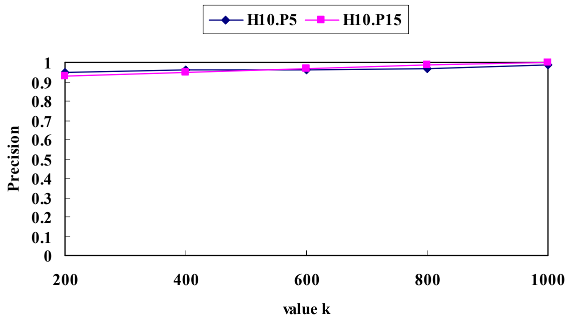


Fig. 12. Performance comparisons of precisions (proportion of the output path traversal patterns are top-*k* path traversal patterns).

structure, **TKP-forest** (a forest of Top-K Path traversal patterns), 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, called **TKP-forest-Maintenance**, is used to guarantee that the upper bound of the summary data structure is predictable. Based on our best knowledge, the proposed DSM-TKP algorithm is the first single-pass algorithm for mining top-*k* path traversal patterns from streaming Web click-sequences. Future work includes mining top-*k* path traversal patterns over stream sliding windows and mining top-*k* path traversal patterns over damped sliding windows.

ACKNOWLEDGEMENTS

The authors thank the reviewers' precious comments for improving the quality of the paper.

REFERENCES

1. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Based*, 1994, pp. 487-499.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2002, pp. 1-16.
3. M. S. Chen, J. S. Park, and P. S. Yu, "Efficient data mining for path traversal patterns," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, 1998, pp. 209-221.
4. Y. L. Cheung and A. W. C. Fu, "Mining association rules without support threshold: with and without item constraints," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, 2004, pp. 1052-1069.
5. A. W. C. Fu, R. W. W. Kwong, and J. Tang, "Mining N-most interesting itemsets," in *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems*, 2000, pp. 59-67.
6. H. F. Li, S. Y. Lee, and M. K. Shan, "DSM-PLW: Single-pass mining of path traversal patterns over streaming web click-sequences," *Computer Networks: Special*

Issue on Web Dynamics, Vol. 50, 2006, pp. 1474-1487.
7. L. Golab and M. T. Ozsu, "Issues in data stream management," *SIGMOD Record*, Vol. 32, 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, 2003, pp. 51-55.
9. A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-*k* elements in data streams," in *Proceedings of the 10th International Conference on Database Theory*, 2005, pp. 398-412.
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 *Proceedings of KDD Workshop on Web Mining as a Premise to Effective and Intelligent Web Applications*, 2003, pp. 71-81.
11. O. Nasraoui, C. Cardona, C. Rojas, and F. González, "TECNO-STREAMS: Tracking evolving clusters in noisy data streams with a scalable immune system learning model," in *Proceedings of the 3rd IEEE International Conference on Data Mining*, 2003, pp. 235-242.
12. P. Tzvetkov, X. Yan, and J. Han, "TSP: Mining top-*k* closed sequential patterns," in *Proceedings of the 3rd IEEE International Conference on Data Mining*, 2003, pp. 347-354.
13. J. Wang, J. Han, Y. Lu, and P. Tzvetkov, "TFP: An efficient algorithm for mining top-*k* frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, 2005, pp. 652-664.

Hua-Fu Li (李華富) received his B.S. degree in Computer Science and Engineering from Tatung Institute of Technology, Taiwan, in 1999, the M.S. degree in Computer Science from National Chengchi University, Taiwan, in 2001, and the Ph.D. degree in Computer Science from National Chiao Tung University, Taiwan, in 2006. He has been an assistant processor in the Department of Computer Science at Kainan University, Taoyuan, Taiwan, since 2007. His research interests include stream data mining, web mining, multimedia data mining and social network mining.

Suh-Yin Lee (李素瑛) received the B.S. degree in Electrical Engineering from National Chiao Tung University, Taiwan, in 1972, the M.S. degree in Computer Science from University of Washington, USA, in 1975, and the Ph.D. degree in Computer Science from Institute of Electronics, National Chiao Tung University. She has been a professor in the Department of Computer Science and Information Engineering at National Chiao Tung University since 1991, and was the chair of that department in 1991-1993. Her research interests include multimedia information system, mobile computing, and data mining.