DSM-PLW: Single-pass mining of path traversal patterns over streaming Web click-sequences ☆

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

[a] *Department of Computer Science and Information Engineering, National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan, ROC*

[b] *Department of Computer Science, National Chengchi University, 64, Sec. 2, Zhi-nan Road, Wenshan, Taipei 116, Taiwan, ROC*

Available online 20 December 2005

Abstract

Mining Web click streams is an important data mining problem with broad applications. However, it is also a difficult problem since the streaming data possess some interesting characteristics, such as unknown or unbounded length, possibly a very fast arrival rate, inability to backtrack over previously arrived click-sequences, and a lack of system control over the order in which the data arrive. In this paper, we propose a projection-based, single-pass algorithm, called DSM-PLW (Data Stream Mining for Path traversal patterns in a Landmark Window), for online incremental mining of path traversal patterns over a continuous stream of maximal forward references generated at a rapid rate. According to the algorithm, each maximal forward reference of the stream is projected into a set of reference-suffix maximal forward references, and these reference-suffix maximal forward references are inserted into a new in-memory summary data structure, called SP-forest (Summary Path traversal pattern forest), which is an extended prefix tree-based data structure for storing essential information about frequent reference sequences of the stream so far. The set of all maximal reference sequences is determined from the SP-forest by a depth-first-search mechanism, called MRS-mining (Maximal Reference Sequence mining). Theoretical analysis and experimental studies show that the proposed algorithm has gently growing memory requirements and makes only one pass over the streaming data.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Web click-sequence streams; Path traversal patterns; Single-pass algorithm

1. Introduction

In recent years, database and knowledge discovery 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*. 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 of such data streams differs from the mining of traditional datasets in the following aspects

☆ Partial results [34] of this study appeared in the Proceedings of the 13th World Wide Web Conference, 2004.

* Corresponding author.

E-mail addresses: hfli@csie.nctu.edu.tw (H.-F. Li), sylee@csie.nctu.edu.tw (S.-Y. Lee), mkshan@cs.nccu.edu.tw (M.-K. Shan).

1389-1286/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2005.10.018

[5,27]: First, each data element in the streaming data should be examined at most once. Second, the memory requirement for mining data streams should be bounded even though new data elements are continuously generated from the data stream. Third, each data element in the stream should be processed as fast as possible. Fourth, the analytical results generated by the online algorithms should be instantly available when user requested. Finally, the frequency of output errors generated by the online algorithms should be constricted to be as small as possible.

As described above, the continuous nature of streaming data makes it essential to use algorithms which require only *one scan* over the stream for knowledge discovery. The unbounded characteristic makes it impossible to store all the data in main memory or even secondary storage. This motivates the design of *summary data structure* with small footprints that can support both one-time and continuous stream queries. In other words, *one-pass* algorithms for mining data streams have to sacrifice the correctness of their analytical results by allowing some counting errors. Hence, traditional *multiple-pass* techniques studied for mining static datasets are not feasible to mine patterns over streaming data.

Recently, some interesting research results have been reported for modeling and computing data streams [31], monitoring statistics over streams [17], and continuous queries over data steams [7]. Furthermore, conventional OLAP (Online Analytical Processing) and data mining models have been extended to tackling data streams, such as multidimensional analysis [12], clustering [28,37,1], and classification [19,32]. Problems related to frequency counting include approximate frequency moments [4], L1 differences [23], synopsis data structure [26], frequent itemsets [10,18,33,9], hot items in dynamic data stream model [16], iceberg queries [22], change mining [21,24,25,36,20], and top-k queries [6]. Algorithms over data streams that pertain to aggregation include approximate quantiles [30] and stream joining [3]. In this paper, we consider a new application of (Web) data stream mining, i.e., *online, single-pass mining path traversal patterns in streaming Web click-sequences*.

The problem of mining *path traversal patterns* from a large *static* Web click dataset was proposed by Chen et al. [11]. Two *multiple-pass* algorithms, FS (Full Scan) and SS (Selective Scan), are proposed. However, the FS and SS algorithms are not feasible for mining the set of path traversal patterns in a streaming environment. Hence, we modified the path traversal pattern mining problem proposed by Chen et al. [11] into a new problem of data stream mining. An efficient, single-pass algorithm, called DSM-PLW (Data Stream Mining for Path traversal patterns in a Landmark Window), is proposed to mine the set of path traversal patterns in the landmark window of an online, continuous stream of Web click-sequences. The purpose of mining patterns in a landmark window of data streams is to discover patterns over the entire history of the data streams [47]. An effective in-memory summary data structure, called SP-forest (Summary Path traversal pattern forest), is proposed for storing the essential information about the frequent reference sequences in the stream so far. Finally, the set of all maximal reference sequences, i.e., path traversal patterns, is determined from the SP-forest by a depth-first-search mining mechanism, called MRS-mining (Maximal Reference Sequence mining). To the best of our knowledge, this is the first study for online, single-pass mining path traversal patterns over streaming Web click-sequences.

The contributions of this paper are described as follows.

- We define a new challenging problem of online, single-pass mining path traversal patterns over streaming Web click-sequences.
- We propose a novel single-pass algorithm, DSM-PLW, to solve this problem efficiently.
- An effective summary data structure, SP-forest, is proposed to maintain the essential information of the stream.

The remainder of the paper is organized as follows. The problem definition and related work are discussed in Section 2. In Section 3, we describe the design of our algorithm for mining path traversal patterns over Web click-sequence streams. Theoretical analysis and performance results are presented in Section 4. Finally, we conclude our work and discuss some future directions in Section 5.

2. Problem definition and related work

2.1. Problem definition

Let S be an infinite sequence of Web clicks, where a Web click wc consists of a 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: $\langle 100, abcde \rangle$, and $\langle 200, abcae \rangle$, where 100, 200 are user 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 with l backward references can be divided into $(l+1)$ maximal forward references. For example, a click-sequence $\langle abcae \rangle$ can be divided into two $MFRs$: $\langle abc \rangle$ and $\langle ae \rangle$, because the second reference a is a backward reference in this click-sequence. Therefore, we can map the problem of mining path traversal patterns into the one of finding frequent occurring consecutive sequences, called **reference sequences** (RSs), among all maximal forward references. The **estimated support** ($esup$) of a reference sequence RS, denoted as $RS.esup$, is the number of maximal forward references in the stream containing RS as a substring. A reference sequence RS is called a **frequent reference sequence** if $RS.esup \geqslant s \cdot N$, where s is a user-defined minimum support threshold in the range of $[0, 1]$, and N is the current length of stream, i.e., the number of maximal forward references so far. A reference sequence $s_1, s_2, \ldots, s_n$, is called a **super-sequence** of another reference sequence $r_1, r_2, \ldots, r_k$ if there exists an i such that $s_{i+j} = r_j$, for $1 \leqslant j \leqslant k$. A frequent reference sequence is called **maximal frequent reference sequence** (abbreviated as **maximal reference sequence** in the context of the paper) if it is not a *substring* of any other frequent reference sequences.

Consequently, the problem of online, single-pass mining path traversal patterns in a landmark window over Web click-sequence streams is to mine maximal reference sequences by *one scan* of a continuous stream of maximal forward references when the value of minimum support threshold s is given.

2.2. Related work

Cooley et al. [15] and Srivastava et al. [44] have surveyed the major technical advances and research problems in Web data mining. In general, Web data mining can be divided into three categories: Web structure mining, Web content mining and Web usage mining. The goal of Web structure mining is to generate a structural summary about the Web site or Web page. The goal of Web content mining is to describe the automatic search of information resource available online, and to discover Web data content. Web usage mining is the process of automatic discovery of user navigation patterns from Web server logs. In this section, a brief review of Web user navigation pattern mining is described as follows.

Chen et al. [11] defined a problem of mining path traversal patterns in a large Web-log dataset. Two algorithms, FS (Full Scan) and SS (Selective Scan), are proposed. These algorithms use level-by-level methods, i.e., apriori-based approaches [1], to discover maximal reference sequences in a static Web click dataset. Although FS and SS mine path traversal patterns in a static Web-log dataset efficiently, they are not feasible for mining streaming Web click-sequences. This is because the FS and SS algorithms need to scan the dataset at least twice.

Spiliopoulou et al. [43] proposed a navigation pattern discovery miner, called WUM (Web Utilization Miner), and proposed an algorithm for building an aggregating tree from static Web logs. Then, WUM mines the Web access patterns by using the MINT mining language. Borges and Levene [8] proposed a model of hypertext that captures the user navigation behavior patterns. The set of user navigation sessions is modeled as a HPG (Hypertext Probabilistic Grammar), and the set of strings which are generated with higher probability correspond to the navigation trials preferred by the user. Pei et al. [39] proposed a WAP-tree (Web Access Pattern tree) to store the frequent Web page-sequences of user navigation behavior, and proposed an efficient pattern-growth WAP-mine algorithm to mine the Web access patterns from the WAP-tree. WAP-mine is a two-pass algorithm. Shan and Li [42] proposed a two-pass algorithm, Fast-Walk, to mine the Web traversal walks. A Web traversal walk is a structural sequence of forward and backward traversal paths. In the Fast-Walk algorithm, an extended prefix-tree structure is constructed in main memory from Web logs, and the frequent Web traversal walks are generated from the in-memory tree structure efficiently.

Pabarskaite [38] suggested several hypotheses that could help improve a Web site's retention and proposed decision trees for Web user behavior analysis. The decision tree package C4.5 is used in [38],

and showed reasonable computational performance and accuracy. Xing and Shen [45] proposed two efficient algorithms, UAM (User Access Matrix) and PNT (Preferred Navigation Tree), based on the selection and time preference concepts for mining user-preferred navigation patterns. Considering the Web site topology, the UAM algorithm can get user access preferred paths by the page–page transition statistics of all users' behaviors. PNT is similar to WAP-tree. However, each node of PNT records the support, which is the frequency and the time of a user's visiting the node along the same route, and the preference represents how users prefer visiting this node from the previous nodes.

Web prefetching and prediction of HTTP requests are important applications of Web usage mining [13,41]. Chen and Chang [13] proposed a popularity-based PPM (Prediction by Partial Match model) for Web prefetching. The popularity-based mode uses grades (grades 3, 2, 1 and 0) to rank URL access patterns and builds these patterns into a predictor tree to aid Web prefetching. The popularity-based PPM uses only the most popular URLs as root nodes and makes space optimizations to the completed tree by removing non-root nodes and those nodes accessed only once. Schechter et al. [41] introduced the use of path profiles for describing HTTP request behavior and proposed an algorithm for creating these path profiles efficiently.

Association rule and sequential pattern mining algorithms are also common for mining Web visitors' behavior [2,29,14,40,35]. Agrawal and Srikant [2] proposed the well-known apriori property, i.e., *all non-empty subsets of a frequent itemset must also be frequent*, and developed three multiple-pass algorithms based on the aprioir property for mining frequent itemsets by using candidate-generation-and-testing approaches. Han et al. [29] proposed a prefix-tree structure FP-tree (Frequent Pattern tree) and a two-pass pattern-growth algorithm FP-growth to discover the set of frequent itemsets without generating candidate itemsets. Chenug and Zaïane [14] proposed a data structure called CATS Tree (Compressed and Arranged Transaction Sequence Tree), an extension of FP-tree, to discover the set of frequent itemsets. The CATS tree is a prefix-tree structure and it contains all elements of the FP-tree including the header, the item links, etc. Pei et al. [40] proposed a two-pass, pattern-growth algorithm PrefixSpan (Prefix-projected Sequential pattern mining) to mine sequential patterns. PrefixSpan finds frequent 1-sequences, i.e., length-1 sequential patterns, after scanning the sequence database once. Then, the database is projected onto smaller datasets according to the frequent 1-sequences. Finally, the set of sequential patterns is found recursively by growing subsequence fragments in each projected database. Although PrefixSpan discovers sequential patterns efficiently, the cost of disk I/O might be high due to the creation and processing of the projected subdatabases. Hence, the two-pass algorithm PrefixSpan is not practical for mining streaming data. Lin and Lee [35] proposed a memory-indexing algorithm MEMISP (MEMory Indexing for Sequential Pattern mining) for fast discovery of sequential patterns. MEMISP reads data sequences into memory in one pass if the memory is large enough to store these sequences. Then MEMISP discovers the sequential patterns by using a recursive find-then-index technique. Although MEMISP is a single-pass algorithm, it is still not feasible for mining patterns in a streaming data. This is because the MEMISP is not an incremental mining algorithm while the data stream is a continuous sequence of data elements.

3. Online single-pass mining streaming Web click-sequences for path traversal patterns: DSM-PLW algorithm

The process of mining path traversal patterns in Web click streams is shown in Fig. 1. Algorithm DSM-PLW (Data Stream Mining for Path traversal patterns in a Landmark Window) is composed of four steps: read a basic window which consists of a fixed sized maximal forward references from the buffer in the main memory (step 1), construct an in-memory summary data structure by processing each incoming basic window (step 2), prune and maintain the summary data structure (step 3), and find the set of path traversal patterns from the current summary data structure (step 4). Steps 1 and 2 are performed in sequence for a new basic window. 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 2–4, and devise algorithms for the effective construction and maintenance of summary data structure, and efficient determination of the set of path traversal patterns.

3.1. Construction of the in-memory summary data structure

In this section, a new in-memory summary data structure, called **SP-forest** (Summary Path traversal

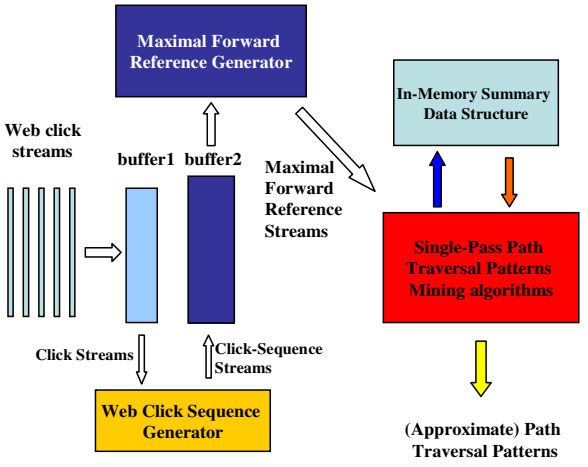

Fig. 1. The process of mining path traversal patterns in Web click streams.

pattern forest), is proposed to store the essential information about path traversal patterns of each incoming basic window, and an efficient algorithm is proposed to construct the summary data structure. Then, we use a running example to illustrate.

Definition 1. A Summary Path traversal pattern forest (abbreviated as **SP-forest**) is a prefix tree-based summary data structure defined below.

1. SP-forest consists of *a list of frequent references* (denoted by **FR-list**), such as $r_1, r_2, \ldots, r_k$, where $r_i.esup \geqslant s \cdot N$, and *a set of Path traversal pattern tree* (abbreviated as **Path-tree**) of references r_i, denoted by r_i.Path-tree, $\forall i = 1, 2, \ldots, k$.
2. Each node in the r_i.Path-tree, $\forall i = 1, 2, \ldots, k$, consists of four fields: *fr_id*, *esup*, *mfr_id*, and *node-link*, where *fr_id* is the *id*entifier of the incoming *f*orward *r*eference, *esup* registers the number of maximal forward references represented by a portion of the path reaching the node with the *fr_id*, the value of *mfr_id* assigned to a new node is the *id*entifier of current *m*aximal *f*orward *r*eference, and *node-link* links up a node with the next node with the same *f_id* in the SP-forest or null id if there is none.
3. Each entry r_i, $\forall i = 1, 2, \ldots, k$, in the FR-list consists of four fields: *fr_id*, *esup*, *mfr_id*, and *head-link*, where *fr_id* registers which forward reference identifier the entry represents, *esup* records the number of maximal forward references in the stream so far containing the reference with identifier *fr_id*, *mfr_id* assigned to a new entry is the identifier of the current maximal forward reference, and *head-link* is a pointer, and points to the root node of the *fr_id*.Path-tree.

Fig. 2 gives the SP-forest construction algorithm. First of all, the DSM-PLW algorithm simply reads a maximal forward reference $MFR_i = \langle r_1, r_2, \ldots, r_j, \ldots, r_m \rangle$ from the buffer to maintain the FR-list. The maintenance process is described as follows. For each reference r_j in MFR_i, if the reference r_j exists in the current FR-list, the estimated support of the reference, i.e., $r_j.esup$, is increased by one. Otherwise, a new entry of form (r_j, 1, i, $\rightarrow r_j$) is created in the FR-list. Note that we use the notation $\rightarrow r_j$ to indicate the *head-link* of r_j, and i is the current MFR's identifier. Next, MFR_i is projected into *m reference-suffix maximal forward references* (denoted by *rs-MFRs*) according to the order of references in the MFR_i. The step is called a *maximal forward reference projection*, and denoted by

Algorithm SP-forest construction

Input: A stream of maximal forward references, MFR_1, $MFR_2, \ldots, MFR_N$, and a user-defined minimum support threshold $s \in (0, 1)$.
Output: An SP-forest so far

```
FR-list = {}; /* initialize the FR-list to empty */
foreach MFR_i = <r_1, r_2,…, r_k> do /* ∀i = 1, 2,…, N, where N is the identifier of current MFR*/
    foreach reference r_j ∈ MFR_i do /* ∀j = 1, 2,…, k */
        if r_j ∉ FR-list then
            create a new entry of form (r_j, 1, i, →r_j) into the FR-list;
        else
            r_j.esup = r_j.esup + 1;
        end if
        call MFR-projection(MFR_i, r_j);
    end for
end for
call SP-pruning(SP-forest, N, s);
```

Fig. 2. Algorithm SP-forest construction.

$MFR\text{-}projection(MFR_i) = \{r_1|MFR_i, r_2|MFR_i, \ldots, r_j|MFR_i, \ldots, r_m|MFR_i\}$, where $r_j|MFR_i = \langle r_j r_{j+1} \cdots r_m \rangle$, $\forall j = 1, 2, \ldots, m$. For example, a maximal forward reference $\langle acdef \rangle$ is projected into five reference-suffix maximal forward references: $\langle acdef \rangle$, $\langle cdef \rangle$, $\langle def \rangle$, $\langle ef \rangle$, and $\langle f \rangle$. Note that the cost of maximal forward reference projection is $(m^2 + m)/2$, i.e., $m + (m-1) + \cdots + 1$. Next, these rs-MFRs with prefix r_i, $\forall i = 1, 2, \ldots, m$, are inserted into the r_i.Path-tree as branches, respectively. If a rs-MFR shares a prefix with a MFR already in the Path-tree, the new MFR will share a prefix of the branch representing that MFR. In addition, an estimated support counter is associated with each node in the Path-tree. The counter is updated when a reference-suffix maximal forward reference causes the insertion of a new branch. Fig. 3 shows the subroutines of SP-forest construction and maintenance.

Example 1. Let the first six maximal forward references in the stream of Web click-sequences be $\langle acdef \rangle$, $\langle abe \rangle$, $\langle cef \rangle$, $\langle acdf \rangle$, $\langle cef \rangle$, and $\langle df \rangle$, where a, b, c, d, e, and f are Web references. The SP-forest with respect to the first two MFRs, $\langle acdef \rangle$ and $\langle abe \rangle$, constructed by DSM-PLW algorithm is shown in Figs. 4 and 5, respectively. Note that the dotted-line arrows, i.e., *node-links*, in Fig. 4 are used to link up a node with the next node with the same *fr_id* in the current SP-forest. However, in the following steps, as demonstrated in Figs. 5–7, the *node-links* are omitted for concise presentation.

First, the DSM-PLW algorithm reads the first maximal forward reference $\langle acdef \rangle$ from the buffer, and projects it into five reference-suffix maximal forward references: $\langle acdef \rangle$, $\langle cdef \rangle$, $\langle def \rangle$, $\langle ef \rangle$, and $\langle f \rangle$. Next, the algorithm inserts $\langle acdef \rangle$, $\langle cdef \rangle$, $\langle def \rangle$, $\langle ef \rangle$, and $\langle f \rangle$ into the empty trees, i.e., a.Path-tree, c.Path-tree, d.Path-tree, e.Path-tree, and f.Path-tree, respectively. The step results in a single path in each Path-tree: $root(a{:}1{:}1) \rightarrow (a{:}1{:}1) \rightarrow (c{:}1{:}1) \rightarrow (d{:}1{:}1) \rightarrow (e{:}1{:}1) \rightarrow (f{:}1{:}1)$, $root(c{:}1{:}1) \rightarrow (c{:}1{:}1) \rightarrow (d{:}1{:}1) \rightarrow (e{:}1{:}1) \rightarrow (f{:}1{:}1)$, $root(d{:}1{:}1) \rightarrow (d{:}1{:}1) \rightarrow (e{:}1{:}1) \rightarrow (f{:}1{:}1)$, $root(e{:}1{:}1) \rightarrow (e{:}1{:}1) \rightarrow (f{:}1{:}1)$, and $root(f{:}1{:}1) \rightarrow (f{:}1{:}1)$. The result is shown in Fig. 4.

Subroutine MFR-projection

Input: A maximal forward reference $MFR_i = <r_1, r_2, \ldots, r_j, \ldots, r_m>$.
Output: r_j.Path-tree, $\forall j = 1, 2, \ldots, m$.

```
foreach reference r_j, ∀j = 1, 2,…,m, in MFR_i do
    call Path-tree-maintenance(r_j|MFR_i, r_j.Path-tree, i);
end for
```

Subroutine Path-tree-maintenance

Input: A reference-suffix maximal forward reference $r_j|MFR_i = <r_j r_{j+1} \ldots r_m>$, r_j.Path-tree, and the identifier of current maximal forward reference i;
Output: A modified r_j.Path-tree, $\forall j = 1, 2, \ldots, m$.

```
foreach reference r_l, ∀l = j, j+1,…,m, in r_j|MFR_i do
    if r_l.Path-tree has a child node with id y such that y.fr_id = r_l.fr_id then
        y.esup = y.esup+1;
    else
        create a new node of form (x_l, 1, i) in the r_l.Path-tree;
    end if
end for
```

Subroutine SP-pruning

Input: A SP-forest, a user-defined minimum support threshold s in the range of [0, 1], and the identifier of current maximal forward reference N.
Output: A SP-forest containing the set of all path traversal patterns.

```
foreach entry r_j ∈ FR-list do
    if r_j.esup < s·N then
        delete r_j.Path-tree;
        delete r_j from FR-list;
        delete the sub-trees of a node whose fr_id is j in other r_l.Path-tree (l ≠ j) by traversing the
            node-links in the SP-forest;
    end if
end for
```

Fig. 3. Subroutines of SP-forest construction algorithm.

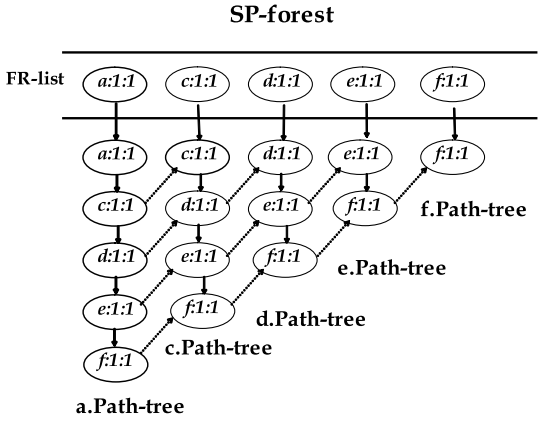

Fig. 4. SP-forest after processing the first maximal forward reference $\langle acdef \rangle$.

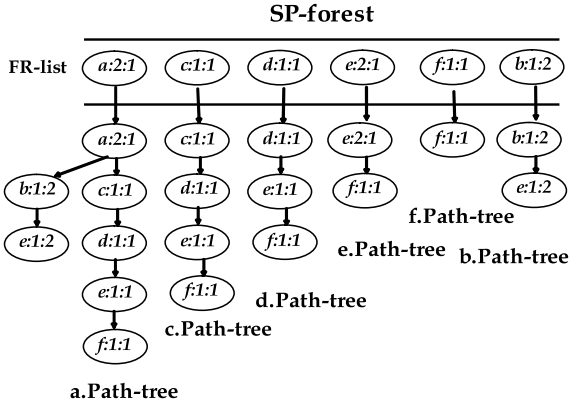

Fig. 5. SP-forest after processing the second maximal forward reference $\langle abe \rangle$.

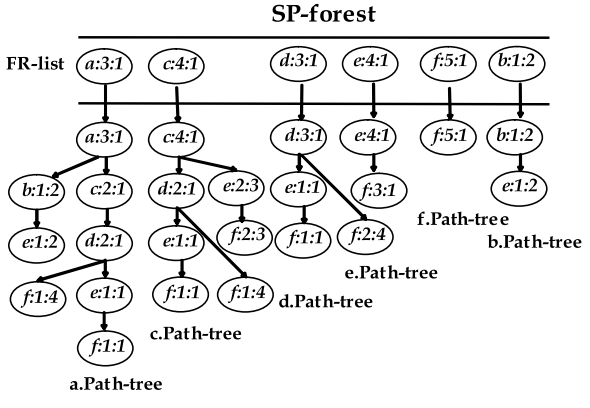

Fig. 6. SP-forest after processing the first six maximal forward references.

Then, DSM-PLW inserts the result of MFR-projection ($\langle abe \rangle$): $\langle abe \rangle$, $\langle be \rangle$, and $\langle e \rangle$ into a.Path-tree,

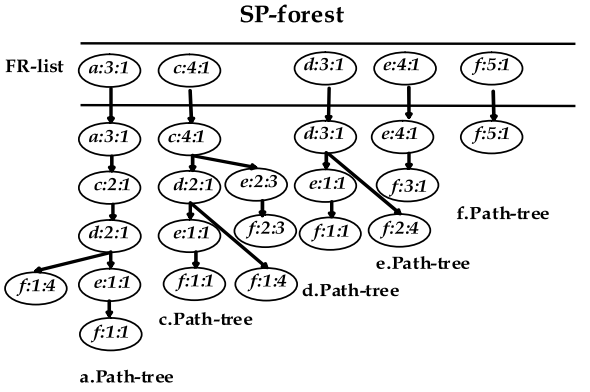

Fig. 7. SP-forest after pruning the infrequent reference b.

b.Path-tree, and e.Path-tree, respectively. Hence, $\langle abe \rangle$ leads to one path with a being the common prefix: $root(a{:}2{:}1) \rightarrow (a{:}2{:}1) \rightarrow (c{:}1{:}1) \rightarrow (d{:}1{:}1) \rightarrow (e{:}1{:}1) \rightarrow (f{:}1{:}1)$ and $root(a{:}2{:}1) \rightarrow (a{:}2{:}1) \rightarrow (b{:}1{:}2) \rightarrow (e{:}1{:}2)$. Then, $\langle be \rangle$ results in a single path in b.Path-tree: $root(b{:}1{:}2) \rightarrow (b{:}1{:}2) \rightarrow (e{:}1{:}2)$. Finally, DSM-PLW algorithm inserts $\langle e \rangle$ into the SP-forest. At this time, no new node is created, but the first path of e.Path-tree is changed to: $root(e{:}2{:}1) \rightarrow (e{:}2{:}1) \rightarrow (f{:}1{:}1)$. The result is shown in Fig. 5. After processing the six maximal forward references, the SP-forest is given in Fig. 6.

3.2. Pruning mechanism of the summary data structure

According to the apriori principle [2], only the frequent references are used to construct candidate k-RSs (k-reference-sequences) in the next pass, where $k > 1$. Thus, the set of candidates containing the infrequent references stored in SP-forest is pruned. The pruning is usually performed periodically or when it is needed.

Let the user-defined minimum support threshold be s in the range of $[0, 1]$, and the length of Web click-sequence stream be N, i.e., N maximal forward references. The pruning mechanism of SP-forest is that a reference sequence X and it super-sequences are deleted from SP-forest if $X.esup < s \cdot N$. For each entry of form (fr_id, $esup$, $mfr_id \rightarrow fr_id$) in the FR-list, if its $fr_id.esup$ is less than $s \cdot (N\text{-}mfr_id + 1)$, it can be regarded as an infrequent reference. At this time, three operations are preformed in sequence. First, DSM-PLW deletes the fr_id. Path-tree. Second, it deletes the reference with id fr_id from the FR-list. Finally, DSM-PLW deletes the infrequent reference with id fr_id and its suffix

paths from other Path-trees by node-links. After pruning all infrequent references from SP-forest, SP-forest contains the set of all frequent path traversal patterns of the stream so far.

Example 2. Let the user-specified minimum support threshold be 0.3. Hence, a reference sequence X is called *infrequent* in Fig. 6 if $X.esup < 0.3 \cdot 6 = 1.8$. At this time, only reference b ($b.esup = 1$) is infrequent by searching the current FR-list. Now, in order to maintain the frequent patterns in the SP-forest, DSM-PLW deletes b.Path-tree, b's suffix paths from a.Path-tree, and b from the FR-list. The result is shown in Fig. 7.

The next step of DSM-PLW algorithm is to determine the set of all path traversal patterns from SP-forest constructed so far. The step is performed only when the analytical results of the stream is requested.

3.3. Determination of path traversal patterns from the summary data structure

Assume that there are k frequent references, namely $r_1, r_2, \ldots, r_k$, in the current FR-list. Let the minimum support threshold be s in the range of $[0, 1]$, and the current length of stream be N. For each entry r_i, $\forall i = 1, 2, \ldots, k$, in the FR-list, DSM-PLW traverses the r_i.Path-tree to find the reference sequences with prefix r_i whose estimated support is greater than $s \cdot N$ in depth-first-search (DFS) manner. Then, DSM-PLW stores the maximal reference sequences in a temporal list, *MRS-list*. Finally, DSM-PLW outputs the set of path traversal patterns stored in the temporal list. Fig. 8 gives the path traversal pattern mining algorithm, called *MRS-mining* (Maximal Reference Sequence mining).

Example 3. The example illustrates the mining of the path traversal patterns from the current SP-forest shown in Fig. 7. Let the minimum support s be 0.3.

First, the MRS-mining algorithm starts the path traversal pattern mining scheme from the first reference a in the FP-list, and generates a frequent reference sequence $\langle acd \rangle$ by DFS. MRS-mining adds $\langle acd \rangle$ into MRS-list because $\langle acd \rangle$ is not a substring of any other patterns stored in the current MRS-list. Next, on the second entry c, MRS-mining algorithm finds two frequent reference sequences: $\langle cd \rangle$ and $\langle cef \rangle$. However, only $\langle cef \rangle$ is added into the MRS-list. This is because $\langle cd \rangle$ is a substring of a generated maximal reference sequence $\langle acd \rangle$. On the third entry d, only one frequent reference sequence $\langle df \rangle$ is generated by MRS-mining, and stored into the MRS-list. On the fourth entry e, only one frequent reference sequence $\langle ef \rangle$ is generated, but it is not a maximal reference sequence. This is because $\langle ef \rangle$ is a substring of $\langle cef \rangle$. On the last entry f, only one frequent reference sequence $\langle f \rangle$ is obtained, but $\langle f \rangle$ is not a maximal reference sequence. This is because $\langle f \rangle$ is a substring of $\langle cef \rangle$. Finally, the MRS-list contains the set of maximal reference sequences, i.e., *path traversal patterns*: $\langle acd \rangle$, $\langle cef \rangle$, and $\langle df \rangle$.

4. Theoretical analysis and performance evaluation

In this section, we discuss the theoretical analysis of space requirements of a prefix tree-based summary data structure, the generation of synthetic path traversal data, and the experimental results

Algorithm MRS-mining (Maximal Reference Sequence mining)

Input: A *SP-forest* constructed so far, the current length of maximal forward references N, and a user-defined minimum support threshold s in the range of [0, 1].
Output: A temporal list of maximal reference sequences, *MRS-list*,

```
MRS-list = ∅;
foreach entry r_i in the current FR-list do
    do Depth-First-Search to find the esup of each reference sequence Y with prefix r_i in
        the r_i.Path-tree;
    if Y.esup ≥ s.N and Y is not a substring of any other frequent reference sequences stored in the
MRS-list then
        add Y into the MRS-list;
    end if
end for
if MRS-list ≠ ∅ then
    output patterns form the MRS-list;
end if
```

Fig. 8. Algorithm of MRS-mining.

of DSM-PLW algorithm on synthetic datasets and real datasets.

4.1. Space upper bound of a prefix tree-based summary data structure

In this section, we discuss the space upper bound of any single-pass algorithm for constructing a prefix tree-based summary data structure.

Theorem 1. *A prefix tree-based summary data structure has at most 2^k nodes for storing the set of all frequent reference sequences of data streams.*

Proof. Let k be the number of frequent references in the stream generated so far. Hence, the number of potential frequent reference sequences is $C(k,1)$ regarding one reference, $C(k,2)$ regarding two references, ..., $C(k,i)$ regarding i references, ..., and $C(k,k)$ regarding k references according to the apriori heuristic. 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. Thus, there are $C(k,1)$ nodes in the first level, $C(k,2)$ nodes in the second level, ..., $C(k,i)$ nodes in the ith level, ..., and $C(k,k)$ nodes in the kth level. There are totally $C(k,1) + C(k,2) + \cdots + C(k,i) + \cdots + C(k,k)$ nodes in the prefix tree-based summary data structure. Consequently, the space upper bound of a prefix tree-based summary data structure is $O(2^k)$. □

4.2. Generation of synthetic traversal paths

To evaluate the performance of DSM-PLW algorithm, two experiments are performed. The experiments were carried out on the synthetic Web traversal path data generator proposed by Chen et al. [11]. We describe it briefly as follows. A traversal tree is constructed to mimic a 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 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, 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 such 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].

Three synthetic data streams, H10P5.D200K, H10P10.D200K, and H10P15.D200K, of size 200,000 reference paths are studied. HxPy means that x is the height of a traversal tree, and y is the average size of the reference paths. D200K means that the number of reference paths is 200,000. A traversal tree for H10 was obtained when the height of the tree is 10, and the fanout at each internal node is

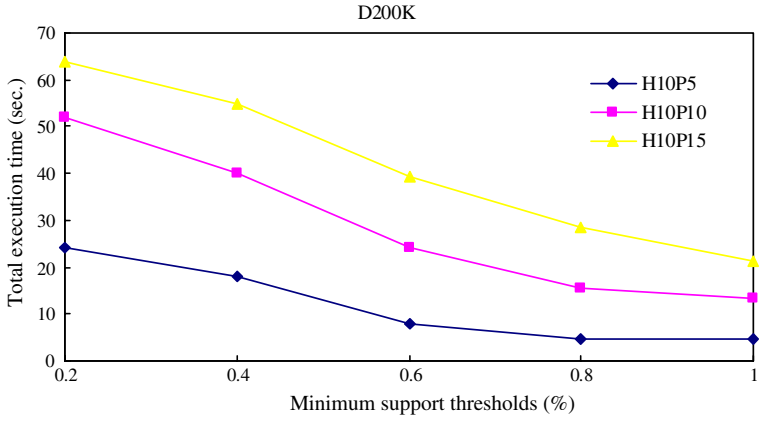

Fig. 9. Performance comparisons of total execution time over various minimum support thresholds.

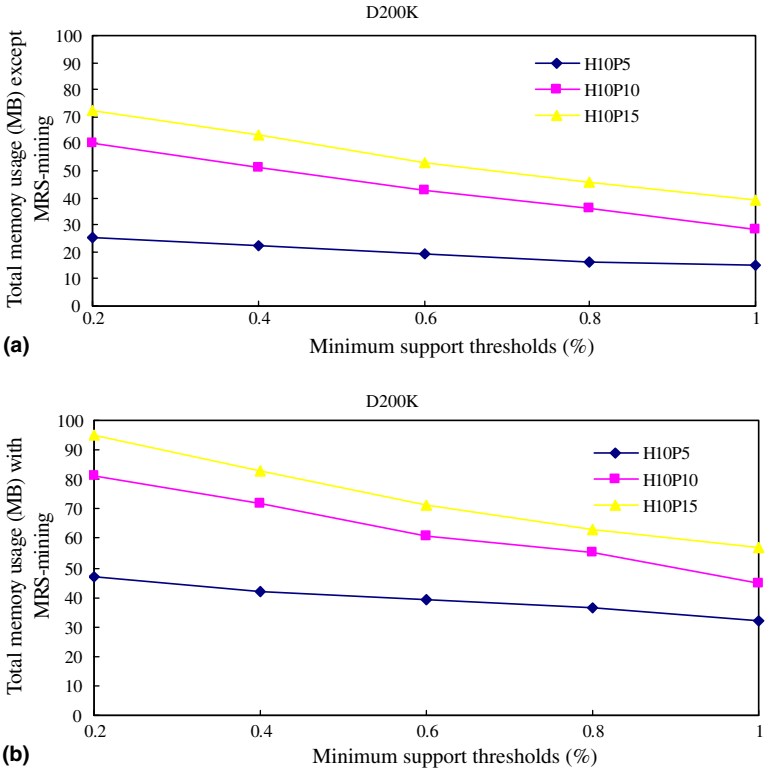

Fig. 10. Performance comparisons of memory usage over various minimum support thresholds: (a) without MRS-mining and (b) with MRS-mining.

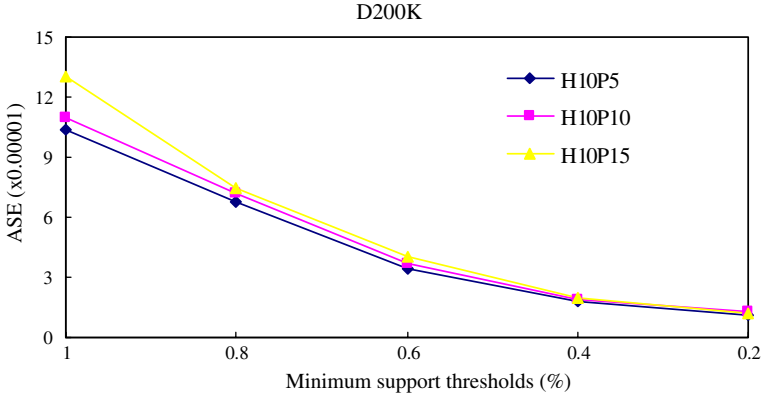

Fig. 11. Accuracy of mining results.

between 4 and 7. The root node consists of seven child nodes. Moreover, the number of internal nodes is 16,200 and the number of leaf nodes is 73,006. In all experiments, the click-sequences of each dataset are looked up in sequence to simulate the environment of a data stream. All the experiments are performed on a 1.80 GHz Pentium 4 processor with 512 megabytes main memory, running on Microsoft Windows 2000. In addition, all the programs are written in Microsoft/Visual C++ 6.0.

4.3. Experimental results of synthetic data

We first evaluated the effect of various minimum support thresholds s for synthetic data streams having a typical value of 200,000 (200K) reference

paths. In Fig. 9, we plot total execution time taken by our algorithm for values of minimum support threshold s ranging from 0.2% to 1%. The figure shows how decreasing s leads to an increase in running time. Fig. 10 shows how decreasing s leads to an increase in memory usage. The memory usage shown in Fig. 10(a) is the memory requirement in Steps 2 and 3 of DSM-PLW algorithm, and Fig. 10(b) is the total memory requirement of DSM-PLW algorithm in Steps 2–4.

To measure the relative accuracy of DSM-PLW algorithm, the *average support error ASE* proposed

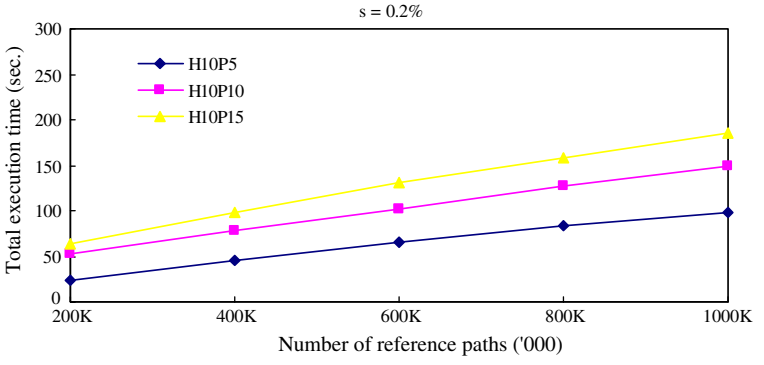

Fig. 12. Linear scalability of the streaming data size.

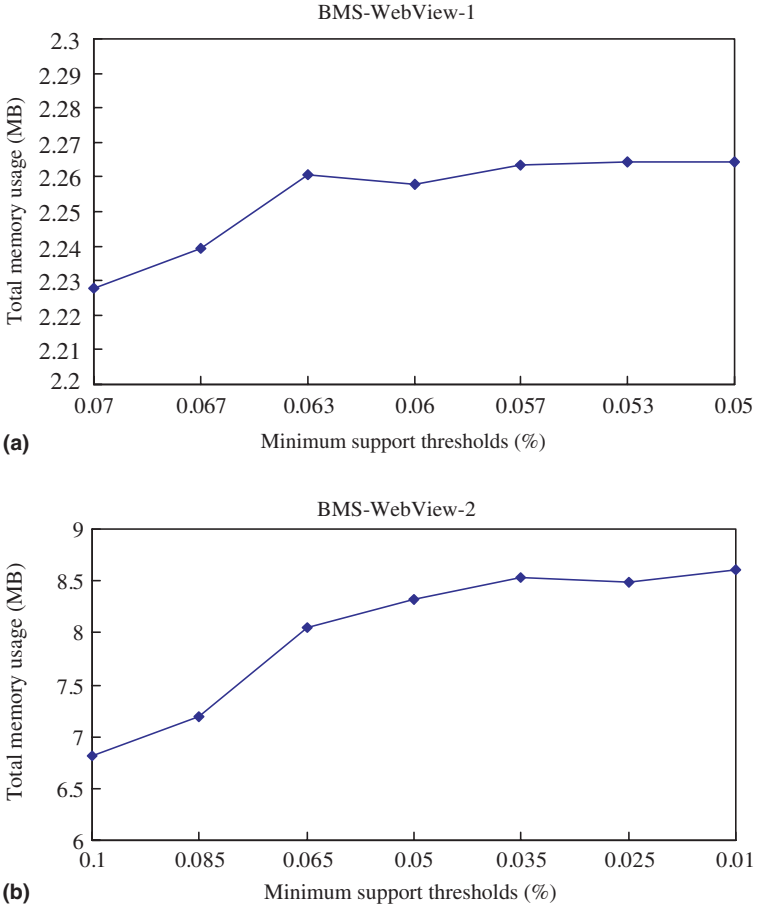

Fig. 13. Memory usage of DSM-PLW on (a) BMS-WebView-1 and (b) BMS-WebView-2 over various minimum support thresholds.

in [9] is used. Fig. 11 shows the average support error of the mining results of the proposed algorithm with respect to that of the FS algorithm [11] performed on the synthetic streaming data by varying the user-specified minimum support threshold s. Generally, the average support error is increased as the value of s is increased in Fig. 11.

To assess the scalability of our algorithm, scale-up experiments were conducted. Fig. 12 shows that the execution time of DSM-PLW increases linearly as the streaming data size increases, ranging from 200K to 1000K. Different minimum support thresholds s yield similar and consistent results. The result of $s = 0.2\%$ is shown in Fig. 12, and it exhibits good linearity in scale-up.

4.4. Experimental results of real data

Two real click-stream datasets, BMS-WebView-1 and BMS-WebView-2, which contain several months worth of click-stream data from two e-commerce Web sites, are used to evaluate the performance of the DSM-PLW algorithm. The real data was provided by Blue Martini Software [46], and is available from the KDD Cup 2000 home page [48]. The BMS-WebView-1 dataset consists of 497 items and 59,602 transactions. The maximum transaction size of BMS-WebView-1 is 267 distinct items and the average transaction size is 2.5 items. The BMS-WebView-2 dataset consists of 3340 distinct items and 77,512 transactions. The maximum transaction size of BMS-WebView-2 is 161 items and the average transaction size is five items. Note that an item is regarded as a reference and a transaction is regarded as a maximal forward reference in these experiments.

In the experiments, two major factors, *memory* and *execution time*, are examined in the online, single-pass mining path traversal patterns of streaming Web click-sequences, since both should be bounded online as time advances. As shown in Fig. 13, the memory usage of DSM-PLW algorithm is relatively insensitive to the minimum support thresholds. As

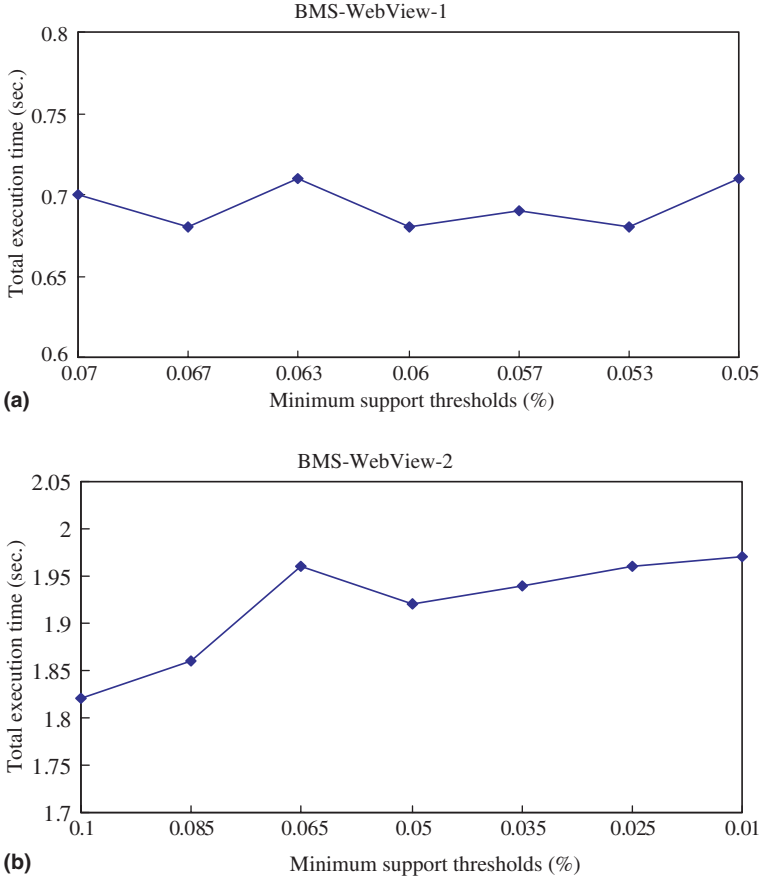

Fig. 14. Execution time of DSM-PLW on (a) BMS-WebView-1 and (b) BMS-WebView-2 over various minimum support thresholds.

the support decreases, the memory consumption of DSM-PLW increases stablely, indicating the feasibility of the proposed algorithm. In Fig. 14, the execution time of DSM-PLW grows smoothly as the support decreases for both real datasets. Hence, the experiments show that DSM-PLW algorithm is a practical method to mine the set of path traversal patterns in real data.

5. Conclusions

This study has presented a new problem of Web data mining, namely, *online single-pass mining path traversal patterns in streaming Web click-sequences*. A new single-pass algorithm the DSM-PLW (Data Stream Mining for Path traversal patterns in a Landmark Window) is proposed to discover the set of all path traversal patterns over the entire history of continuous stream of Web click-sequences. In the DSM-PLW algorithm, an effective in-memory summary data structure SP-forest (Summary Path traversal pattern forest) is developed to maintain the essential information of all maximal reference sequences in the stream so far. The set of all maximal reference sequences, i.e., path traversal patterns, is determined from the SP-forest by a depth-first-search mechanism, called MRS-mining (Maximal Reference Sequence mining). Theoretical analysis and experimental results show that the DSM-PLW algorithm can meet the performance requirements of data stream mining, namely, *single-pass*, *bounded space*, and *real time*. Future work includes single-pass mining top-k path traversal patterns and mining path traversal patterns in a tuple-based or time-based sliding window that contains the most recent N maximal forward references.

Acknowledgements

The authors are grateful to three anonymous referees whose valuable comments helped to improve the content of this paper. We would like to thank Blue Martini Software for contributing the KDD Cup 2000 data. The research is supported in part by the National Science Council, Project No. NSC 93-2213-E-009-043, Taiwan, Republic of China.

References

[1] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for clustering evolving data streams, in: Proc. VLDB, 2003, pp. 81–92.

[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proc. VLDB, 1994, pp. 487–499.

[3] N. Alon, P. Gibbons, Y. Matias, M. Szegedy, Tracking join and self-join sizes in limited storage, in: Proc. PODS, 1999, pp. 10–20.

[4] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, in: Proc. STOC, 1996, pp. 20–29.

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

[6] B. Babcock, C. Olston, Distributed top-k monitoring, in: Proc. ACM SIGMOD, 2003, pp. 28–39.

[7] S. Babu, J. Widom, Continuous queries over data streams, SIGMOD Rec. 30 (3) (2001) 109–120.

[8] J. Borges, M. Levene, Data mining of user navigation patterns, in: Proc. WEBKDD, 1999, pp. 92–111.

[9] J.H. Chang, W.S. Lee, Finding recent frequent itemsets adaptively over online data streams, in: Proc. ACM SIGKDD, 2003, pp. 487–492.

[10] M. Charilar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: Proc. ICALP, 2002, pp. 693–703.

[11] M.-S. Chen, J.-S. Park, P.S. Yu, Efficient data mining for path traversal patterns, IEEE Trans. Knowl. Data Eng. (TKDE) 10 (2) (1998) 209–221.

[12] Y. Chen, G. Dong, J. Han, B.W. Wah, J. Wang, Multi-dimensional regression analysis of time-series data streams, in: Proc. VLDB, 2002, pp. 323–334.

[13] X. Chen, X. Zhang, A popularity-based prediction model for web prefetching, IEEE Comput. 36 (3) (2003) 63–70.

[14] W. Cheung, O.R. Zaïane, Incremental mining of frequent patterns without candidate generation or support constraint, in: Proc. IDEAS, 2003, pp. 111–116.

[15] R. Cooley, B. Mobasher, J. Srivastava, Web mining: information and pattern discovery on the World Wide Web, in: Proc. ICTAI, 1997, pp. 558–567.

[16] G. Cormode, S. Muthukrishnan, What's hot and what's not: tracking most frequent items dynamically, ACM Trans. Database Syst. 30 (1) (2005) 249–278.

[17] M. Datar, A. Ginois, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows, in: Proc. SODA, 2002, pp. 635–644.

[18] E. Demaine, A. López-Ortiz, J.I. Munro, Frequent estimation of internet packet streams with limited space, in: Proc. ESA, 2002, pp. 348–360.

[19] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proc. ACM SIGKDD, 2000, pp. 71–80.

[20] G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang, P.S. Yu, Online mining of changes from data streams: research problems and preliminary results, in: Proc. ACM SIGMOD MPDS, 2003.

[21] G. Dong, J. Li, Efficient mining of emerging patterns: discovering trends and differences, in: Proc. ACM SIGKDD, 1999, pp. 43–52.

[22] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Moteani, J.D. Ullman, Computing iceberg queries efficiently, in: Proc. VLDB, 1998, pp. 299–310.

[23] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan, An approximate L1-difference algorithm for massive data streams (extended abstract), in: Proc. IEEE FOCS, 1999, pp. 501–511.

[24] V. Ganti, J. Gehrke, R. Ramakrishnan, A framework for measuring changes in data characteristics, in: Proc. PODS, 1999, pp. 126–137.
[25] V. Ganti, J. Gehrke, R. Ramakrishnan, Mining data streams under block evolution, SIGKDD Explor. 3 (2) (2002) 1–10.
[26] P.B. Gibbons, Y. Matias, Synopsis data structures for massive data sets, in: Proc. SODA, 1999, pp. 909–910.
[27] L. Golab, M.T. Ozsu, Issues in data stream management, SIGMOD Rec. 32 (2) (2003) 5–14.
[28] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams, in: Proc. FOCS, 2000, pp. 359–366.
[29] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: a frequent-pattern tree approach, Data Min. Knowl. Discovery 8 (1) (2004) 53–87.
[30] J.M. Hellerstein, P.J. Haas, H. Wang, Online aggregation, in: Proc. ACM SIGMOD, 1997, pp. 171–182.
[31] M.R. Henzinger, P. Raghavan, S. Rajagopalan, Computing data streams, Technical Report 1998-011, Digital Eqipment Corporation, Systems Research Center, May 1998.
[32] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proc. ACM SIGKDD, 2001, pp. 97–106.
[33] R. Karp, C. Paradimitriou, S. Shenker, A simple algorithm for finding elements in sets and bags, ACM Trans. Database Syst. (TODS) 28 (1) (2003) 51–55.
[34] H.-F. Li, S.-Y. Lee, M.-K. Shan, On mining webclick streams for path traversal patterns, in: Proc. WWW, 2004, pp. 404–405.
[35] M.-Y. Lin, S.-Y. Lee, Fast discovery of sequential patterns through memory indexing and database partitioning, J. Inform. Sci. Eng. (JISE) 21 (1) (2005) 109–128.
[36] B. Liu, W. Hsu, H.-S. Han, Y. Xia, Mining changes for real-life applications, in: Proc. DaWaK, 2000, pp. 337–346.
[37] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, R. Motwani, Streaming-data algorithms for high-quality clustering, in: Proc. ICDE, 2002, pp. 685–696.
[38] Z. Pabarskaite, Decision trees for web log mining, Intell. Data Anal. 7 (2) (2003) 141–154.
[39] J. Pei, J. Han, B. Mortazavi-Asl, H. Zhu, Mining access patterns efficiently from Web logs, in: Proc. PAKDD, 2000, pp. 396–407.
[40] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: the PrefixSpan approach, IEEE Trans. Knowl. Data Eng. 16 (10) (2004) 1424–1440.
[41] S. Schechter, M. Krishnan, M.D. Smith, Using path profiles to predict HTTP requests, Comput. Networks 30 (1–7) (1998).
[42] M.-K. Shan, H.-F. Li, Fast discovery of structure navigation patterns from web user traversals, in: Proc. SPIE DMKD, 2002, pp. 272–283.
[43] M. Spiliopoulou, L.C. Faulstich, K. Winkler, A data miner analyzing the navigational behaviour of web users, in: Proc. ACAI, 1999, pp. 588–589.
[44] J. Srivastava, R. Cooley, M. Deshpande, P.-N. Tan, Web usage mining: discovery and applications of usage patterns from web data, SIGKDD Explor. 1 (2) (2000) 12–23.
[45] D. Xing, J. Shen, Efficient data mining for web navigation patterns, Information and Software Technology 46 (1) (2004) 55–63.
[46] Z. Zheng, R. Kohavi, L. Mason, Real world performance of association rule algorithms, in: Proc. ACM SIGKDD, 2001, pp. 401–406.
[47] Y. Zhu, D. Shasha, StatStream: statistical monitoring of thousands of data streams in real time, in: Proc. VLDB, 2002, pp. 358–369.
[48] http://www.ecn.purdue.edu/KDDCUP/.

Hua-Fu Li received the B.S. degree in Computer Science and Engineering from Tatung Institute of Technology and the M.S. degree in Computer Science from National Chengchi University, in 1998 and 2000, respectively. He is currently working towards the Ph.D. degree in the Department of Computer Science and Information Engineering at National Chiao-Tung University. His current research interests include data mining, multimedia systems and bioinformatics.

Suh-Yin Lee 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.

Man-Kwan Shan received the B.S. degree in computer engineering and the M.S. degree in computer and information science both from National Chiao-Tung University, Taiwan, in 1986 and 1988, respectively. From 1988 to 1990, he served as a lecture in the Army Communications and Electronics School. Then, he worked as a lecture at the Computer Center of National Chiao-Tung University, where he supervised the Research and Development Division. He received the Ph.D. degree in Computer Science and Information Engineering from National Chiao-Tung University in 1998. Then he joined the Department of Computer Science at National Chengchi University as an assistant professor. He became an associated professor in 2003. His current research interests include data mining, multimedia systems and bioinformatics.