Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007)

On Mining Dynamic Web ClickStreams for Frequent Traversal Sequences

Jiadong Ren, Xiaojian Zhang, and Huili Peng
College of Information Science and Engineering, Yanshan University, Qinhuangdao, 066004, China
xiaojian007love@gmail.com

Abstract-**Although frequent traversal sequence (FTS) mining has been extensively studied over the last decade in web usage mining, it is challenging to extend the mining technique to dynamic web click streams. The main challenge is that existing false-positive methods control memory consumption and output accuracy by a relaxation ratio *r* (*r* = *e*/*s*, *e* is the error parameter, and *s* is the specified minimum support). However, the higher the value of *r*, the more saving is the memory consumption and the better recall but degrades the output precision, while on the contrary, decreasing *r* gives a more precise output but needs higher storage space. In this paper, the upper and lower bounds are established to constrain *r*, a weighted harmonic average (*WHA*) of the two bounds is designed to adjust *r*, and a novel algorithm *FTS-Stream* is proposed to find the FTS over a time-sensitive sliding window. Thus, the precision and recall can be maintained with the *WHA* (*r*). Our analysis and experiments show that *FTS-Stream* has high accuracy and requires less memory in dynamic Web clickstreams.**

I. INTRODUCTION

Mining frequent traversal sequence (FTS) has been studied over the last decade [1, 2], which is an important application of sequential mining technique for mining traversal patterns. Past research only focuses on mining FTS from static database. Recent emerging applications, such as network traffic analysis, Web click stream mining, sensor network data analysis, and dynamic tracing of stock fluctuation, call for study of a new kind of data, called data streams, as opposed to finite, statically stored data sets. Traditional Web click stream mining focuses on off-line data mining. However, in practice, Web click stream are generate in the form of continuous, rapid data steams, and then stored in web servers. Therefore, mining dynamic Web click stream is more important in some web applications, such as on-line monitoring use behavior, on-line performance analysis, and on-line improving web connectivity etc.

There exist many algorithms for mining frequent pattern (FP) over data streams, such as *Lossy Counting* [3], *estWin* [7], and *DSM-PLW* [6] etc. Most of these algorithms utilize a relaxation ratio, *r* (*r*=*e*/*s*, *e* is the error parameter, and *s* is the specified minimum support), to control the output quality of the FP. Therefore, these algorithms are mainly false positive, the output will plunge into a dilemma because of *r*. A smaller *r* can present a more accurate output but worsen the recall, lower the processing efficiency, and generate a larger number of patterns. On the contrary, a higher *r* can save the memory consumption and better recall rate but degrade the output precision. The false negative algorithms *MineSW* [9] and *FDPM* [12] are proposed to deal with the problem caused by *r*. However, the algorithms also encounter the former problem, since the two algorithms do not adopt the constraint strategy. The research of mining FP in data streams can be divided into three fields: landmark windows model, titled-time windows model, and sliding windows model, as described briefly as follows. Manku and Motwani [3] firstly proposed the landmark model, which utilize the entire history data between a particular point of time and the current time for mining. Giannella et al. [8] developed the titled-time model that mines the recent data at a fine granularity while mining the long-term data at a coarse granularity. Teng et al. [5] proposed the sliding windows model, which gives a window size *w*, only the latest transactions are utilized for mining. That is to say, as a new transaction has been reached, the oldest transaction in the sliding window is expired.

Generally, patterns embedded in data streams are more likely to be change as time goes by. Identifying the recent change of data streams can quickly provide valuable information for the analysis of the data streams. Thus, in certain applications, users can only be interested in the data recently arriving within a fixed time period. For example, when mining the Web click streams, the most recent data usually provides more useful information than those that arrived previously. Obviously, landmark and titled-time window models are unable to satisfy this need. On the contrary, the sliding window model achieves the goal. In this paper, we develop a novel algorithm, *FTS-Stream*, for mining FTS from dynamic Web click Streams based on a time-sensitive sliding window model. J.Han et al. [4] introduced that data mining is an interactive process, and users should directly take part in the process through query language or GUI. Therefore, according to J.Han's idea, we design an efficient constraint strategy, which users can give two decent bound parameters to constrain the relaxation ratio, *r*. Although the strategy possibly limits the frequency of some traversal sequences, we can discover more interesting FTS. To solve the problem caused by *r*, we propose a weighted harmonic count, and design a weighted harmonic average of the two bounds parameters to replace *r*. Our experiments show that our algorithm can simultaneously maintain precision and recall of the output, obtain highly precise mining results, and consume less main memory.

We summarized the contributions of this paper. Firstly, a constrained methodology is introduced for mining dynamic Web click streams. Next, we propose an effective summary data structure, *IPFTS-tree* (Improved Prefix Frequent Traversal Sequences tree), to maintain the essential information of the Web click streams. Thirdly, we develop a novel single-pass algorithm, *FTS-Stream*, to build and maintain *IPFTS-tree* to mine the FTS over a time-sensitive sliding window model.

The remaining of the paper is organized as follow. Section 2 presents the related work and the problem definition is given in

Section 3. Section 4 introduces the constraint strategy and harmonic count. Section 5 presents the *FTS-Stream* algorithm, while Section 6 introduces the experiments. Section 7 draws a conclusion.

II. RELATED WORK

Mining FP from data streams has been investigated by many researchers. Existing streaming algorithms mainly focus on landmark window model [3, 6]. However, most of these algorithms adopt an unchanged granularity, that is landmark model is not aware of time and therefore can not distinguish old data and new ones. In many cases, FP are usually time sensitive, and the old FP may have lost their attraction and importance. To overcome this difficulty, many approaches based on sliding window model are proposed. These approaches mainly care the changes and trends of the recent data. Manku, Chang and Lee [3, 13] propose the *Lossy Counting* algorithm and *Carma* algorithm, which adopt estimation mechanism to mine an approximate set of the FP. Lee et al. [10] propose a method to mine FP from the candidate 2-itemsets for each slide. But their approach may generate huge candidate itemsets, which consume large storage space. *Moment* algorithm proposed by Chi et al. [11]. Their algorithm is not suitable for mining FTS since *Moment* mainly finds closed FP. Yu et al. [12] utilize the theory of *Chernoff bound* to propose a false negative algorithm. Their method uses a predefined threshold to control the bound of memory usage and the quality of output. Cheng et al. [9] also propose a false negative algorithm, *MineSW* with a progressively increasing minimum support function. Although the two false negative methods can solve some questions that the false positive methods exist, they may not tackle the dilemma caused by *r*. All the previous works only consider a fixed number of transactions as the basic unit, which is not easy for people to specify. By contrast, it is natural for people to specify a time period as the basic unit. Therefore, in this paper, we propose the time-sensitive sliding window model, which regards a fixed time period as the basic unit for mining.

III. FTS-STREAM ALGORITHM

A. Problem Definition

Let $P = \{P_1, P_2, \ldots, P_n\}$ be the complete set of web pages. A session, *S*, is a traversal sequence that is ordered by timestamp in Web click data. A traversal sequence $ts = <P_1, P_2, \ldots, P_m>$ ($P_i \in P$, $1 \leq i \leq m$) is a list of web page which is ordered by traversal time, and each web page can repeatedly appear in *ts*. Consider two traversal sequences $ts_1 = <a_1, a_2, \ldots, a_n>$ and $ts_2 = <b_1, b_2, \ldots, b_m>$ ($n \leq m$). If there exists integers $1 \leq i_1 < i_2 < \ldots \leq m$ with $a_1 = b_{i1}$, $a_2 = b_{i2}, \ldots, a_n = b_{im}$, then ts_1 is a subsequence of ts_2, and ts_2 is a super-sequence of ts_1. We write a $ts = <P_1, P_2, \ldots, P_m>$ as $ts = <P_1 P_2 \ldots P_m>$ in this paper.

Given a Web click stream *Wcs*, Utilizing the cube model proposed in [15], *Wcs* is converted into traversal sessions, which compose the Session streams $Ss = \{S_1, S_2, \ldots, S_m, \ldots\}$, where, S_i denotes a session in *Ss*. In this paper, we adopt the Session streams instead of original Web click streams to mine the FTS over the time-sensitive sliding window model.

Given a time point *t* and a time period *tp*, the set of all the sessions arriving in [*t*-*tp*+1, *t*] will form a basic block. A Session stream *Ss* is decomposed into a sequence of basic blocks, which are assigned with serial numbers. Given a window with length *w*, we slide it over those basic blocks to observe a set of overlapping blocks, where each block sequence is called the time-sensitive sliding window (abbreviated as *TSsw*). A *TSsw* in the session streams is a window that slides forward for every basic block.

A time interval in the *Ss* is a set of successive basic block units, denoted as $B = <B_i, \ldots, B_j>$, where $i \leq j$. We define B_i as the current basic block unit, within which a variable number of sessions may arrive and $|B_i|$ as the number of session in B_i. For each current block B_i, $TSsw_i$ consists of the $|w|$ consecutive basic blocks from B_{i-w+1} to B_i. The $TSsw_i$ is denoted as $TSsw_i = <B_{i-w+1}, \ldots, B_i>$. We define *Session* (*B*) as the set of sessions that arrive on Web click streams in a time interval *B*, and |*Session* (*B*)| as the number of sessions in *Session* (*B*). The *count* of traversal sequence *ts* over *B*, denoted as *Count* (*ts*, *B*), is the number of sessions in *Session* (*B*) that include *ts*. Given a user predefined minimum support threshold, s ($0 \leq s \leq 1$), *ts* is a FTS over *B* if $Count\ (ts, B) \geq s\ |Session\ (B)|$. Consequently, the problem of online, single-pass mining FTS in a *TSsw* over a session stream *Ss* is to mine the set of FTSs by one scan of a continuous stream of sessions when *s* is given.

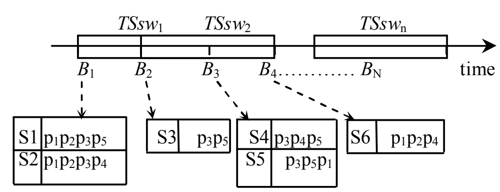


Fig. 1. Sessions in two TSsws

Example Given an example to show the definition, Fig. 1 gives six sessions that are recorded the four basic block units. The four block units form two successive windows, $TSsw_1 = <B_1, B_2, B_3>$ and $TSsw_2 = <B_2, B_3, B_4>$. Let the minimum support count be 3. We can get the set of FTS over $TSsw_1$ and $TSsw_2$, which are $\{<p_1>, <p_3>, <p_5>, <p_3p_5>\}$ and $\{<p_3>, <p_5>, <p_3p_5>\}$.

B. Constraint Strategy and Weighted Harmonic Count

1) Constraint Strategy

The precision, recall, and efficiency of mining FTS in dynamic Web click streams environment are close relaxed to three factors: the constraint strategy, the method to recording the history information, and the summary data structure. We firstly describe the constraint strategy as follows.

Given the nature of the Web click streaming data, there exist two sources of error when estimating frequent traversal sequences. One is that it is possible that some traversal sequences observed as frequent might in fact not be frequent anymore from a longer observation of the Web click stream. The other is that some traversal sequences observed as not frequent may well in fact be frequent from a longer history of the Web click stream. The first error source is precision-oriented, and the second is recall-oriented. Existing algorithms use $r=e/s$ to control the accuracy of the mining result. However, r can lead to a problem introduced in former sections. Therefore, we give two limited parameters λ_1, λ_2 to constrain the r, and take two efficient strategies to meet the mining purpose of the users.

(1) If $r<\lambda_1$, where λ_1 is the lower bound parameter, then the second error source will be triggered. This case denotes a smaller r not only degrades the accuracy of the recall-oriented output, but increases the main memory consumption, and lowers the processing efficiency. Thus the relaxation ratio r should be larger than λ_1.

(2) If $r>\lambda_2$, where λ_2 is the upper bound parameter, then the first error source will be triggered. This case denotes a larger r gives a bad precision-oriented output. That is, larger r will degrade the mining precision. Thus the relaxation ratio r should be smaller than λ_2.

2) Weighted Harmonic Count

Minimizing r ($r\approx\lambda_1$) can lead to minimize the first error source, but lower the mining efficiency and maximize the second source of error. On the other hand, Maximizing r ($r\approx\lambda_2$) leads to minimize the second error source, but maximize the first error source. Therefore, in this paper, we proposed a weighted harmonic average (abbreviated as *WHA*) of λ_1 and λ_2, to replace the relaxation ratio r. Thus, we can adjust the importance of one error source against the other by adjusting the ξ value. That is, ξ is a regulatory factor, which function mainly tackles the problem caused by the relaxation ratio r.

$$WHA\ (r) = (1+\xi^2)\ \lambda_1\lambda_2/(\lambda_1 + \xi^2\lambda_2)\ . \tag{1}$$

$$E = s\times WHA\ (r) = s \times (1+\xi^2)\ \lambda_1\lambda_2/(\lambda_1 + \xi^2\lambda_2). \tag{2}$$

E in the equality (2) does not equal e ($r=e/s$), since r has been replaced by the *WHA* (r). Based on the equalities (1) and (2), the potential count of a ts over a basic block B_i is defined as follows:

$$Count\ (ts, B_i) = \begin{cases} 0 & \text{if } Count\ (ts, B_i) < E\ |Session\ (B_i)| \\ Count\ (ts, B_i) & \text{otherwise.} \end{cases} \tag{3}$$

Thus, the support count of ts over a time interval $B=<B_j,\ldots,B_m>$ is defined as follows. The type of support count is called accumulated count.

$$Count\ (ts, B) = \sum Count\ (ts, B_i),\ B_i\in B=<B_j,\ldots,B_m>\ . \tag{4}$$

In this way, each ts is associated with the potential count and accumulated count. Moreover, the sum of the two counts is regarded as the count of the ts in $TSsw_i$.

Given parameters λ_1, λ_2, and let $TSsw_i = < B_{i-w+1},\ldots,B_i>$ be a current time-sensitive sliding window, its size is w, and $TS_R = < B_{i-R+1},\ldots,B_i>$, where $1\le R\le w$, be the most recent R basic block units in $TSsw_i$. TS_R is a subsequence of $TSsw_i$. The size of TS_R is $|Session\ (TS_R)|$. We define a weighted harmonic count (denoted as *WHC* ()) as follows.

$$WHC\ (R) = \lceil s\ |Session\ (TS_R)| \times WHA\ (r)\rceil. \tag{5}$$

If we maintain all possible traversal sequences in the current sliding window $TSsw_i$, this will require too much space, so we only maintain the most recent R basic block units, only keep the FTS in the window, and drop the remaining tail sequences of $TSsw_i$. Obviously, the ts over $<B_{i-w+1},\ldots,B_{i-R}>$ is considered as infrequent traversal sequences, and its potential count will be taken as 0. Specially, we drop the tail $<B_{i-w+1},\ldots,B_{i-R}>$ when the following condition holds:

$$\sum Count\ (ts, B) < E \sum|Session\ (B_i)|, \text{ where } B_i\in B=<B_{i-w+1},\ldots,B_{i-R}>\ . \tag{6}$$

3) FTS-Stream

Our algorithm *FTS-Stream* (Frequent Traversal Sequence in dynamic Web clickStream) is composed of four steps: read a fixed size w window Session streams in the memory (step1), construct an in-memory summary data structure *IPFTS-tree* (Improved Prefix Frequent Traversal Sequence tree) by processing each incoming basic block unit B_i (step2), prune and maintain the summary data structure (step3), and mine the set of FTS from the current *IPFTS-tree* (step4). Steps 1 and 2 are performed in traversal sequence for a new sliding window *TSsw*. Steps 3 and 4 are usually performed periodically or when they are needed. Since the step 1 is straightforward, we shall focus on steps 2 and 4, devise algorithms for the effective construction and maintenance of data structure, and determination of the set of FTS. The process of mining frequent traversal sequences in Web click streams is shown Fig. 2.

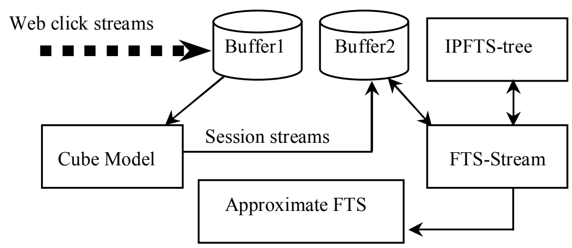


Fig. 2. The process of Mining FTS

a. IPFTS-tree Construction

An improved prefix frequent traversal sequence tree (abbreviate as *IPFTS-tree*) is a based on prefix tree data structure defined as below.

(1) *IPFTS-tree* consists of one root labeled as "Root", and a set of page-prefix subtrees (potentially FTS with its subset) as the children of the root.

(2) Each node in the tree consists of six fields: *page*, *MAC*, *flag*, *Bid*, *P.count*, and *A.count*, where *page* registers which is the last web page of ts, *MAC* registers the number of sessions represented by the portion of the path reaching this node, *flag* registers whether the node is updated in current basic block unit B_i, *Bid* registers which is the id of the basic block unit, B_{Bid}, where ts is inserted into the *IPFTS-tree*, *P.count*, and *A.count* register prudential count and accumulated count respectively.

(3) Each node in the *IPFTS-tree* represents a FTS (from root to this node), and its support is equal to the support of the node. Thus, an *IPFTS-tree* is similar with *WAP-tree* [2], but their structures are different. The difference is that *IPFTS-tree* stores

FTS instead of Web click streams. Fig. 3 shows the structure of the *IPFTS-tree*.

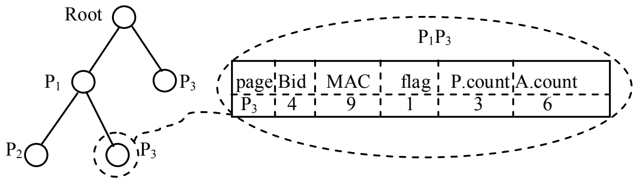


Fig. 3. Structure of the IPFTS-tree

b. FTS-Stream Algorithm

Algorithm 1 CIPFTS-Tree: Construct *IPFTS-tree*
Input: A Session stream *Ss*, *s*, *WHA* (*r*), *w*, λ_1 and λ_2;
Output: An *IPFTS-tree*;
Create the root of an *IPFTS-tree*, *T*, and label it with "Root";
For (each $B_i \in TSsw_{first}$=<$B_1, \ldots, B_j$>, ($1 \leq i \leq j$))
  Mine all FTS over Session (B_i);
  For (each $ts \in$ FTS) // $TSsw_{first}$ denotes the first window.
    If ($ts \not\subset T$) and (*Count* (ts, B_i) $\geq E$ |*session* (B_i)|)
      Create a new node of form (page, *i*, 1, 1, 0);
      Bid (*ts*) =*i*; *Count* (*ts*) = *Count* (*ts*, B_i);
    If (($ts \subset T$)
      Add *Count* (*ts*, B_i) to *Count* (*ts*);
    If (*Count* (*ts*) < *WHC* (*i-Bid* (*ts*) +1))
      Delete *ts* from *T*;
      Stop mining the supersets of *ts* over session (B_i);
      Call MIPFTS-tree;

Subroutine 1 MIPFTS-tree: Maintain *IPFTS-tree*
Input: An *IPFTS-tree* structure, *s*, λ_1, λ_2, *w*, an incoming B_i.
Output: The updated *IPFTS-tree*.
For (each incoming $B_i \in TSsw$)
  Mine all FTSs over session (B_i);
  For (each $ts \in$ FTS)
    If ($ts \subset T$) and ($flag \neq 1$)
      Add *Count* (*ts*, B_i) to *Count* (*ts*);
      Call PIPFTS-tree;
    If ($ts \not\subset T$) and (*Count* (*ts*, B_i) $\geq E$ |*session* (B_i)|)
      Create a new node of form (page, *i*, 1, 1, 0);
      Bid (*ts*) = *i*; *Count* (*ts*) = *Count* (*ts*, B_i);
  For (each expiring $B_{i-w+1} \in TSsw$)
    If ($ts \subset T$) and (*i-Bid* (*ts*) + 1 $\geq w$)
      Count (*ts*) = *Count* (*ts*) - *Count* (*ts*, B_{i-w+1});
    If (*Count* (*ts*) =0)
      Delete *ts* from *T*;
    Else *Bid* (*ts*) =*i-w*+2;

Subroutine 2 PIPFTS-tree: prune *IPFTS-tree*
Input: An *IPFTS-tree* structure, *s*, λ_1, λ_2, *w*.
Output: The *IPFTS-tree* containing the set of FTS.
For (each $ts \in$ FTS)
  If ((*i-Bid* (*ts*) + 1< *w*) and (*Count* (*ts*) < *WHC* (*i-Bid* (*ts*) +1))) or ((*i-Bid* (*ts*) + 1) $\geq w$) and (*Count* (*ts*) <*WHC* (*w*))
    Delete *ts* from *T*;
    Delete the sub-trees of a node whose *Bid* is *i* by traversing the *IPFTS-tree*;

Algorithm 2 FTS-Stream
Input: A Session stream *Ss*, *s*, *WHA* (*r*), λ_1, λ_2 and *w*.
Output: A temporal list of FTS, FTS-list.
FTS-list =$\varnothing$;
Scan a B_i, and collect all FTS;
Call MIPFTS-tree;
Do depth-first-search to mine the FTS;
If (*Count* (*ts*) $\geq s$ |*session* (*w*)|)
  Store *ts* in the FTS-list;
If (FTS-list$\neq \varnothing$)
  Output FTS from the FTS-list;

FTS-Stream algorithm for mining FTS over a time-sensitive sliding window is described in Algorithm 2. In the window *TSsw* initialization phase, an *IPFTS-tree* is created and all FTS are stored in the tree. After the *TSsw* becoming full, we begin to slide *TSsw*. That is, a new basic block unit is appended to the *TSsw*, and the expiring block unit is removed from the window. In this phase, FTS and *IPFTS-tree* are maintained. When a new basic block unit B_i arrives, we mine *ts* from the B_i and update the *IPFTS-tree* structure. For each *ts*, if *ts* does not appear in the *IPFTS-tree* and *Count* (*ts*, B_i) $\geq E$ |session (B_i)|, then we insert *ts*. If *ts* appears, we add *Count* (*ts*, B_i) to *Count* (*ts*) and check the *flag* label of the node with *ts*. If $flag \neq 1$, then we update the node, and check whether *ts* should be removed from the *IPFTS-tree* or not. If *ts* meet the condition of subroutine 2, we remove *ts* from the tree structure. When an old basic unit B_{i-w+1} expires, we should check the weighted harmonic count of *ts*, which is counted from B_{i-w+1}. If *i-Bid* (*ts*) + 1$\geq w$, then we can subtract the support count of *ts*. For traversal sequence *ts* in the *IPFTS-tree*, if its weighted harmonic count is less than a pruning threshold, it is pruned from the *IPFTS-tree*. Finally, we output all the FTS whose support is greater than the minimum support. Besides, let *k* be the number of FTS in the Web click stream generated so far. An *IPFTS-tree* structure has at most 2^k nodes for storing the set of all FTSs of Web click streams.

IV. Experiment Results

Our algorithm was written in C++ and compiled using gcc. All of our experiments are performed on a 2.4GHz Pentium IV processor with 512 MB of main memory, 768 MB of virtual memory, and running on Redhat 9.0. We pursue the experiments on real datasets to evaluate the performance of *FTS-Stream* algorithm. The 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. The real datasets was provided by Blue Martin Software [14], and is available form the KDD Cup 2000 home page. The BMS-WebView-1 consists of 367 distinct pages, 59602 sessions and the average session size contains 7-13 pages. The BMS-WebView-2 consists of 320 distinct pages, 537083 sessions and the average session size is ten pages. Each *TSsw* consists of 20 basic block units, and each basic unit includes 100k sessions. To evaluate the performance of *FTS-Stream*, three group experiments are performed.

A. Two Bounds Constraint

In the first group experiment, we run the *FTS-Stream* algorithm from tow different aspects. One aspect, *FTS-Stream* does not contain the two bound parameters λ_1 and λ_2. The other aspect, *FTS-Stream* contains the two bound parameters. We set λ_1 =0.001, λ_2 =0.999, ξ =0.031, and s=0.01, respectively. Let ET1 be the execution time without λ_1 and λ_2, and ET2 be the execution time with λ_1 and λ_2. Let PT (PT=1-(ET2/ET1)) be the improved performance in percentage. Fig. 4 shows that PT of the *FTS-Stream* clearly increases with the datasets changing from 200k to 1000k.

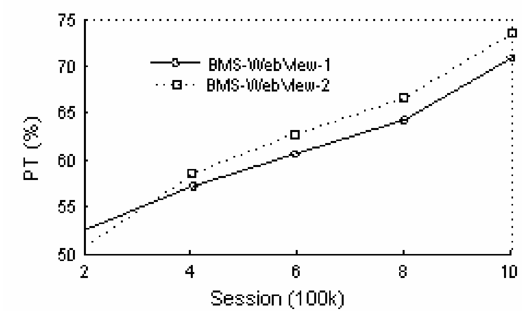
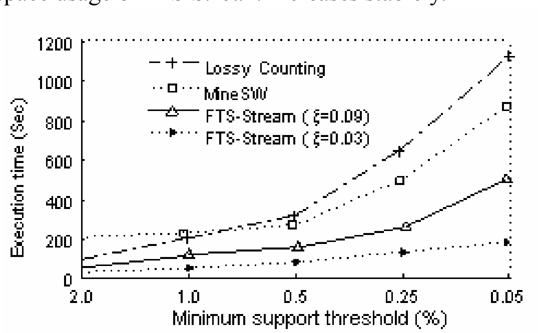


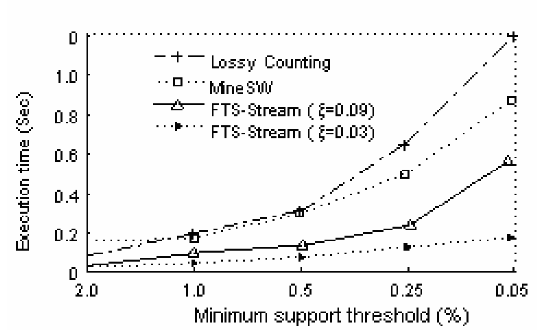
Fig. 4. Improved performances in percentage for FTS-Stream

B. Changing Minimum Support threshold

In the second group experiment, we test the performance of *FTS-Stream* by comparing with previous algorithms *Lossy Counting* and *MineSW*. However, we renew to set the value of the regulatory factor, ξ=0.03 and ξ=0.09. We still hold the values of support threshold s and the two bounds λ_1 and λ_2 as same as the first group experiment. We measure *FTS-Stream* with *Lossy Counting* and *MineSW* in four aspects: execution time, precision, recall and space usage. The results can be seen in Fig. 5 ~ Fig.8. In Fig. 5, the execution time of *FTS-Stream* grows smoothly as the support threshold decrease from 2.0% to 0.05%. However, when ξ=0.03, the time of *FTS-Stream* is over 3 times faster than ξ=0.09, while *FTS-Stream* (ξ=0.03) is about 4 times than faster *MineSW*, and 10 more times over *Lossy Counting*. Fig. 6 and Fig. 7 show the precision and recall comparison among several algorithms with the changing of the support. In this situation, *FTS-Stream* behaves best. When ξ=0.09, precision and recall of *FTS-Stream* are over 0.93, while ξ=0.03, the two aspects are about 0.98.Through adjusting the regulatory factor, precision and recall can cater to the purpose of the users. As shown in Fig. 8, the space usage of *FTS-Stream* is relatively insensitive to the support. As the support decreases, the space usage of *FTS-Stream* increases stablely.

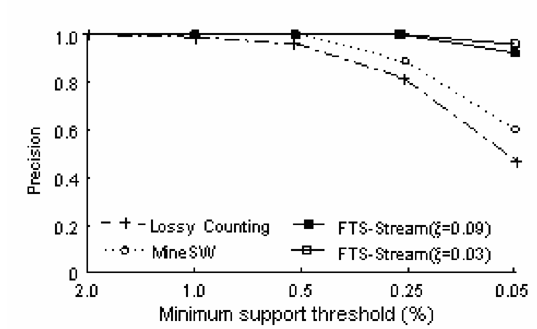


(a) BMS-WebView-1

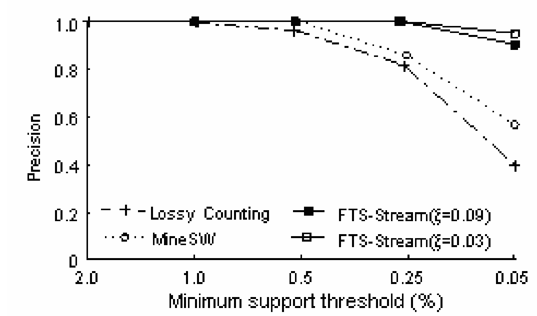


(b) BMS-WebView-2

Fig. 5. Execution time on two datasets with different threshold

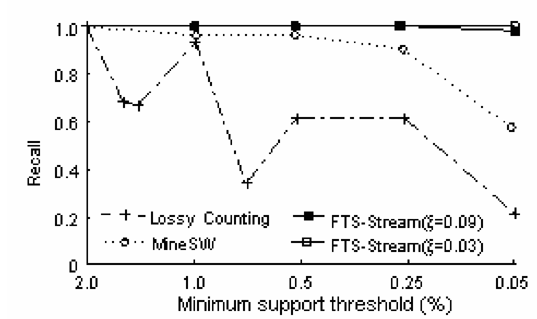


(a) BMS-WebView-1

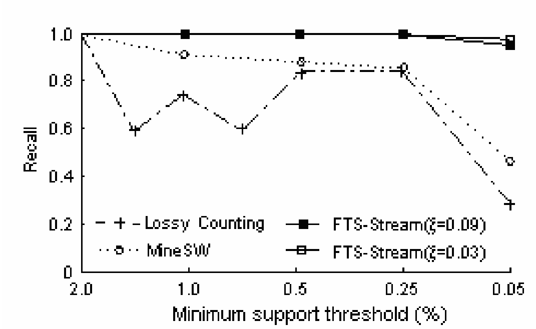


(b) BMS-WebView-2

Fig. 6. Precision on different threshold

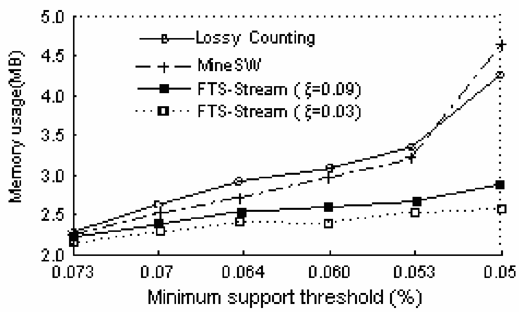


(a) BMS-WebView-1

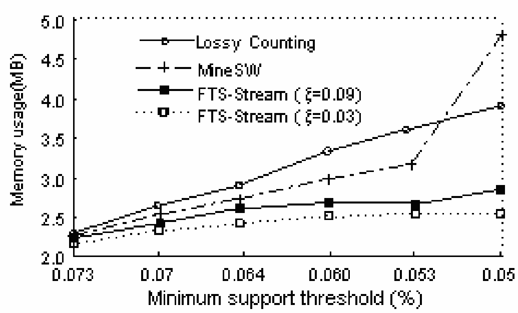


(b) BMS-WebView-2

Fig. 7. Recall on different threshold



(a) BMS-WebView-1

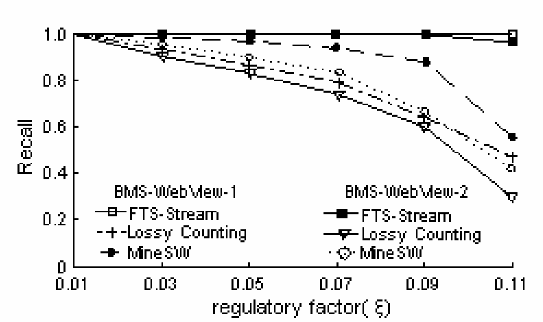


(b) BMS-WebView-2

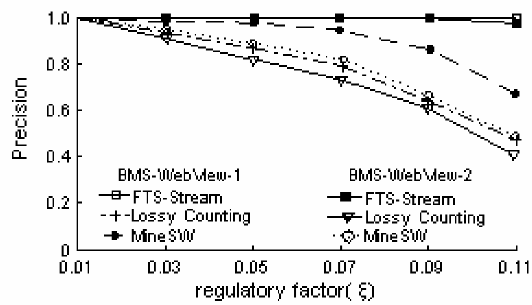
Fig. 8. Memory Usage over different threshold

C. Adjusting the Regulatory Factor

In the third group experiment, we measure the recall and precision of *FTS-Stream* with *Lossy Counting* and *MineSW* by adjusting the value of the regulatory factor. λ_1, λ_2 and s hold the same values as the former experiments. In Fig. 8, we plot the recall and precision of our algorithm for values of ξ ranging from 0.01 to 0.11. The figure shows how increasing ξ leads to decrease in recall and precision. Fig. 9 (a) and (b) show that *FTS-Stream* almost has 100% recall and precision as ξ increases from 0.01 to 0.11. However, the precision and recall of *Lossy Counting* and *MineSW* sharply drop. The result indicates that *Lossy Counting* and *MineSW* often reckon on the r to control the recall and precision of the output, while *FTS-Stream* adopt constraint strategy to limit r, and utilize the weighted harmonic average of the two bounds to replace it. As a result, we can avoid the problem caused by r.



(a) Recall on different regulatory factor



(b) Precision on different regulatory factor

Fig. 9. Recall and Precision on different threshold

V. CONLUSIONS

In this paper, we propose a novel constraint-based algorithm *FTS-Stream* to discover the set of frequent traversal sequences over a time-sensitive sliding window. An effective in-memory summary data structure *IPFTS-tree* is developed to maintain the essential information of FTS in the Web click streams so far. In the *FTS-Stream* algorithm, the weighted harmonic average with a constraint strategy is used to tackle the abuse of the relaxation ratio r. When the lower bound and upper bound are set, we can adjust the value of regulatory factor to get the decent recall and precision on the mining results. The experimental results show that our algorithm significantly outperforms the known *Lossy Counting* and *MineSW* algorithms in terms of execution time, recall, precision, and memory consumption.

REFERENCES

[1] M.S. Chen, J.S, Park, and P.S. Yu: Efficient Data Mining for Path Traversal Patterns. In *IEEE Trans. Knowl. Data Eng*, Vol. 10, No. 2, pp. 209-221, 1998.

[2] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu.: Mining Access Pattern Efficiently from Web Logs. In *Proc. of PAKDD*, 2000, pp. 396-407.

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

[4] Jiawei Han, Micheline Kamber. Data Mining: Concepts and Techniques. In *K. Morgan Kanfmann, Chinese,* 2001.

[5] W. G. Teng, M. S. Chen, and P. S. Yu.: A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In *Proc. of VLDB*, 2003.

[6] Hua-Fu Li, Suh-Yin Lee, and Man-Kwan Shan.: DSM-PLW: Single-Pass Mining of Path Traversal Patterns over Streaming Web Click-Sequences. In Journal of Computer Networks, Vol. 9, No. 19, pp. 126-142, 2005.

[7] J. H. Chang and W. S.Lee. estWin.: Online Data Stream Mining of Recent Frequent Itemsets by Sliding Window Method. In *Journal of Information Science*, Vol. 31, No. 2 2005.

[8] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu.: Mining Frequent Patterns in Data Streams at Multiple Time Granularities. H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), *Next Generation Data Mining,* 2003, pp. 191-212.

[9] J. Cheng, Y. Ke, and Wilfred NG.: Maintaining Frequent Itemsets over High-Speed Data stream. In *Proc. of PAKDD*, 2006, pp. 462-467.

[10] C. Lee, C. Lin, and M. Chen. Sliding-window Filtering: an Efficient Algorithm for Incremental Mining. In *Proc*. of *CIKM*, 2001.

[11] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz. Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. In *Proc. of ICDM*, 2004, pp. 59-66.

[12] J. Yu, Z. Chong, H. Lu, and A. Zhou.: False positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In *Proc. of VLDB*, 2004.

[13] C. Hidber. Online Association Rule Mining. In *Proc. of SIGMOD*, 1999, pp. 145-156.

[14] Z. Zheng, R. Kohavi, and L. Mason.: Real World Performance of Association Rule Algorithm. In *Proc. of ACM SIGKDD*, 2001, pp. 401-406.

[15] Q. Yang, J. Huang, and M. Ng.: A Data Cube Model for Prediction-Based Web Prefetching. In *Journal of Intelligent Information System*, Vol. 20, No. 6, 2003, pp. 11-30.

REFERENCES

[1] M.S. Chen, J.S, Park, and P.S. Yu: Efficient Data Mining for Path Traversal Patterns. In *IEEE Trans. Knowl. Data Eng*, Vol. 10, No. 2, pp. 209-221, 1998.

[2] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu.: Mining Access Pattern Efficiently from Web Logs. In *Proc. of PAKDD*, 2000, pp. 396-407.

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

[4] Jiawei Han, Micheline Kamber. Data Mining: Concepts and Techniques. In *K. Morgan Kanfmann, Chinese,* 2001.

[5] W. G. Teng, M. S. Chen, and P. S. Yu.: A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In *Proc. of VLDB*, 2003.

[6] Hua-Fu Li, Suh-Yin Lee, and Man-Kwan Shan.: DSM-PLW: Single-Pass Mining of Path Traversal Patterns over Streaming Web Click-Sequences. In Journal of Computer Networks, Vol. 9, No. 19, pp. 126-142, 2005.

[7] J. H. Chang and W. S.Lee. estWin.: Online Data Stream Mining of Recent Frequent Itemsets by Sliding Window Method. In *Journal of Information Science*, Vol. 31, No. 2 2005.

[8] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu.: Mining Frequent Patterns in Data Streams at Multiple Time Granularities. H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), *Next Generation Data Mining,* 2003, pp. 191-212.

[9] J. Cheng, Y. Ke, and Wilfred NG.: Maintaining Frequent Itemsets over High-Speed Data stream. In *Proc. of PAKDD*, 2006, pp. 462-467.

[10] C. Lee, C. Lin, and M. Chen. Sliding-window Filtering: an Efficient Algorithm for Incremental Mining. In *Proc*. of *CIKM*, 2001.

[11] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz. Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. In *Proc. of ICDM*, 2004, pp. 59-66.

[12] J. Yu, Z. Chong, H. Lu, and A. Zhou.: False positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In *Proc. of VLDB*, 2004.

[13] C. Hidber. Online Association Rule Mining. In *Proc. of SIGMOD*, 1999, pp. 145-156.

[14] Z. Zheng, R. Kohavi, and L. Mason.: Real World Performance of Association Rule Algorithm. In *Proc. of ACM SIGKDD*, 2001, pp. 401-406.

[15] Q. Yang, J. Huang, and M. Ng.: A Data Cube Model for Prediction-Based Web Prefetching. In *Journal of Intelligent Information System*, Vol. 20, No. 6, 2003, pp. 11-30.