ISSN:0975-9646

Comparison of Different Navigation Prediction Techniques

Prasad J. Koyande, Kavita P. Shirsat

Computer Engineering, Mumbai University
Vidyalankar Institute of Technology
Wadala, Mumbai, India

Abstract --- **Discovering web navigation patterns plays an important role in web mining which is used for prediction and management of website efficiently. As we all know that web site structure is always changed, many Navigation Prediction Techniques need not only consider the frequency of click behavior but also web site structure to mine web navigation patterns for navigation prediction, dynamic mining approach is also based on the previous mining results and formed new patterns just from the modifying some part of the web data. We had compared Different Navigation Prediction Techniques based on criteria's like data structure used for intermediate storage and mining, original database used for mining, re-mining is done to track changes in the web structure, pattern tree construction for improving prediction phase and Technique used for navigation prediction and we found that Mining Web Navigation Patterns with Dynamic Thresholds for Navigation Prediction is the best from that technologies**

Keywords— **Web mining, Navigation Prediction**

I. Introduction

In recent years, vast amount of data would be easily produced and collected from the web environment because of the growth of Internet. Hence, how to discover the useful information and knowledge efficiently from these vast amounts of web data has become a more and more important topic recently. Web navigation pattern mining is present to improve the web services by extracting useful information and knowledge from vast amount of web data.

However, the web data grows rapidly and some of the user sequences may be out of date over time. Therefore, the web navigation patterns should be updated when the new web data is inserted into original web navigation sequence database. Therefore, the web navigation patterns should be updated when the new web data are inserted into original web navigation sequence database. In addition, the frequent web navigation patterns may not always be the interesting web navigation patterns because the access design of web pages is not the same. Therefore, a flexible model of mining navigation patterns with dynamic thresholds is needed. Unfortunately, to the best of our knowledge, there is no any web navigation pattern mining algorithm which considers dynamic thresholds. Besides, there is no any existing work could make navigation prediction in the incremental mining environment.

In this paper, we study a novel efficient incremental algorithm for mining web navigation patterns with dynamic thresholds named id-Matrix-Miner (Inverted-database Matrix Miner), for re-mining all the web navigation patterns when the database is updated. Furthermore, a novel structure we proposed named id-Matrix (Inverted-database Matrix) and inverted file database are selected as our storage structure for id-Matrix-Miner to store original web dataset. The id-Matrix is kept in memory and the projected database is store in hard disk. That is, all of the necessary information in original web dataset is stored in these two kinds of storage. Hence, id-Matrix-Miner can avoid unnecessary scanning of the original web dataset. Thus, I/O cost could be frugal. The id-Matrix-Miner is about 3-5 times faster than IDTM algorithm in most case. Moreover, based on the id-Matrix-Miner, we also study a navigation prediction model which could handle such incremental mining environment.

II. Navigation pattern mining

Web navigation pattern has a great resemblance to web traversal pattern. The most significant difference between the two kinds of patterns is that web navigation patterns consist of contiguous web page references. Thus, the traversal pattern mining algorithms are easy to apply to the navigation pattern mining. Most of studies had discussed incremental web traversal or navigation patterns mining problems which consider the uniform threshold and dynamic threshold.

A. FS-Miner Algorithm

1) *Frequent Sequences:* Let $I = \{i1, i2, ..., im\}$ be a set of unique items, such as page references. A sequence **Seq** = $<p1p2...pn>$ is an ordered collection of items with $pi \in I$ for $1 \leq i \leq n$. A database **DB** (for web usage mining typically a web log file) stores a set of records (sessions). Each record has two fields: the record ID field, **SID**, and the input sequence field, **InSeq**.[8]

For a link h, the *support count*, *Supplink*(h), is the number of times this link appears in the database. For example if the link $a-b$ appears in the database five times we say that $Supplink(a - b) = 5$. For a sequence $Seq = <p1p2...pn>$ we define its size as n which is the number of items in that sequence. Given two sequence $S = <p1p2...pn>$ and $R = <q1q2...qm>$ we say that S is a subsequence of R if there is some i, $1 \leq i \leq m - n + 1$, such that $p1 = qi$, $p2 = qi+1$, ..., $pn = qi+(n-1)$.

The support count *Suppseq*(*Seq*) for a sequence *Seq* is the number of times the sequence appears in the database either as a the full sequence or as a subsequence of sessions.

The behavior of our system is governed by two main parameters. The first parameter is *minimum* ***link*** *support count*, *MSuppClink*, which is the minimum count that a link should satisfy to be considered potentially frequent. *MSuppClink* is obtained by multiplying the total number of links in the database by a desired minimum link support threshold ratio *MSuppRlink*. *MSuppRlink* is the frequency of the link in the database to the total number of links in the database (*Supplink*/total # of links in the database) which a link has to satisfy in order to be considered potentially frequent. *MSuppRlink* is a system parameter (not set by
the user) and is used by the FS-tree construction algorithm to decide what links to include in the FS-tree as will be discussed later. The second parameter *MSuppCseq*, is the ***minimum sequence support count***, that denotes the minimum number of times that a sequence needs to occur in the database to be considered frequent. *MSuppCseq* is obtained by multiplying the total number of links in the database by a desired minimum sequence support threshold ratio *MSuppRseq*. This desired ratio is the frequency of the sequence in the database to the total number of links in the database (*Suppseq*/total # of links in the database) which a sequence has to satisfy in order to be considered frequent. *MSuppRseq* is set by the user and is used by the FS-Mining algorithm during the mining process.

Based on *MSuppClink* and *MSuppCseq* we classify the links in the database into three types:

• ***Frequent links***: links with support count *Supplink* ≥*MSuppCseq*≥ *MSuppClink*. These links are stored in *HT* and are represented in the FS-tree and can be part of frequent sequences.

• ***Potentially Frequent links***: links with support count *Supplink* ≥*MSuppClink* and *Supplink* <*MSuppCseq*. These links are stored in the *HT* and are represented in the FS-tree but they can't be part of frequent sequences (needed for efficient incremental and interactive performance).

• ***Non-frequent links***: links with support count *Supplink* <*MSuppClink*. These links are stored in *NFLT* and are not represented in the FS-tree (needed for efficient incremental and interactive performance).

Only frequent links may appear in frequent sequences, hence, when mining the FS-tree we consider only links of this type. Before we introduce the FS-mine algorithm, we highlight the properties of the FS-tree.

2) *Properties of the FS-trees:* The FS-tree has the following properties that are important to the FS-mine algorithm:

• Any input sequence that has non-frequent link(s) is pruned before being inserted into the FS-tree.

• If *MSuppClink* <*MSuppCseq*, the FS-tree is storing more information than required for the current mining task.

• We can obtain all possible subsequences that end with a given frequent link *h* by following the *ListH* pointer of *h* from the header table to correct FS-tree branches.

• In order to extract a sequence that ends with a certain link *h* from an FS-tree branch, we only need to examine the branch prefix path that ends with that link (*h*) backward up to (maximum) the tree root.

Now we describe in detail the mining steps that we use to extract frequent sequences from the FS-tree. We assume *MSuppClink* = 2 and *MSuppCseq* = 3 as our running example.

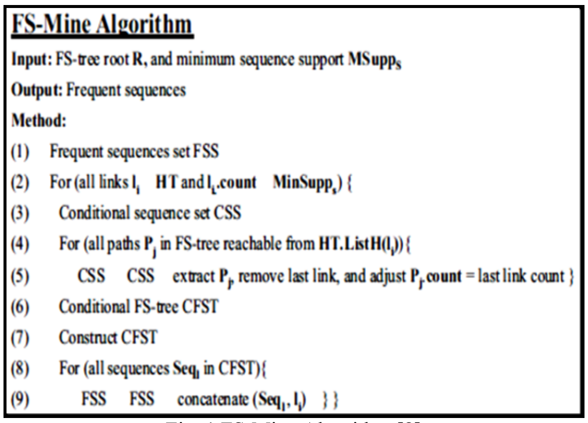
3) *FS-tree Mining Step:*. Fig 1 lists the FS-Mine Algorithm. The algorithm has four main steps that are performed for only frequent links (potentially frequent links are excluded) in the header table (*HT*):

- *Extracting derived paths.* For link *h* in*HT* with *Supplink*(*h*) ≥*MSuppCseq* we extract its derived paths by following the *ListH* pointer of *h* from *HT* to edges in the FS-tree. For each path in the FS-tree that contains *h* we extract its path prefix that ends at this edge and go maximum up to the tree root7. We call these paths *derived paths* of link *h*. For example, from Fig. 2, if we follow the *ListH* pointer for the link $e - h$ from the header table we can extract two derived paths: ($c - d$: 4, $d - e$: 4, $e - h$: 1) and ($b - d$: 3, $d - e$: 2, $e - h$: 2).

- *Constructing conditional sequence bas:.* Given the set of derived paths of link *h* extracted in previous step we construct the *conditional sequence base* for *h* by setting the frequency count of each link in the path to the count of the *h* link (this gives the frequency of the full derived path). We also remove *h* from the end of each of the derived paths For example, given the two derived paths extracted above for link $e - h$, the conditional base for that link consists of: ($c - d$: 1, $d - e$: 1) and ($b - d$: 2, $d - e$: 2).

- *Constructing conditional FS-tree.* Given the conditional base for *h*, we create a tree and insert each of the paths from the conditional base of *h* into it in a backward manner. We create necessary nodes and edges or share them when possible (incrementing edges counts). We call this tree the *conditional FS-tree* for link *h*. For example, given the conditional base for link $e - h$ the constructed conditional FS-tree is shown in Fig. 2.

- *Extracting frequent sequences:* Given a *conditional FS-tree* of a link *h*, we perform a depth first traversal for that tree and return only sequences satisfying *MSuppCseq*. By traversing the conditional FS-tree of link $e - h$ only the sequence <*de*> satisfies the *MSuppCseq*, so we extract it. We then append the link $e{-}h$ to the end of it to get the full size frequent sequence: <*deh* : 3> where 3 represents the support (count) of that sequence.



```
FS-Mine Algorithm
Input: FS-tree root R, and minimum sequence support MSupp_S
Output: Frequent sequences
Method:
(1) Frequent sequences set FSS
(2) For (all links l_i   HT and l_i.count   MinSupp_s) {
(3)   Conditional sequence set CSS
(4)   For (all paths P_j in FS-tree reachable from HT.ListH(l_i)) {
(5)     CSS   CSS   extract P_j, remove last link, and adjust P_j.count = last link count }
(6)   Conditional FS-tree CFST
(7)   Construct CFST
(8)   For (all sequences Seq_l in CFST){
(9)     FSS   FSS   concatenate (Seq_l, l_i)   } }
```

Fig. 1 FS-Mine Algorithm [8]

We perform the same steps for the other frequent links in *HT*, namely $d - g\ a - b$, $b - c$, $d - e$, and $c - d$. The detailed mining steps for these links are shown in Table 1. The last column in that table gives the final result for the mining process. The generated frequent sequences are: <*deh* : 3>, <*abc* : 4>, <*cde* : 4>, and <*bcd* : 3>in addition to the frequent links themselves: (<*eh* : 3>, <*dg* : 4>, <*ab* : 4>, <*bc* : 5>, <*de* : 6>, and <*cd* : 7>) as they are considered frequent sequences of size 2.

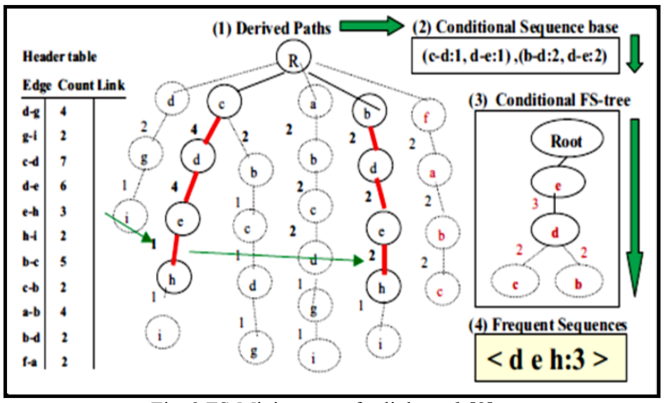


Fig. 2 FS-Mining steps for link $e - h$ [8]

4) *Incremental Web Traversal Pattern Mining*

In order to mine the web traversal patterns incrementally, we use the previous mining results to discover new patterns such that the mining time can be reduced. Therefore, how to choose a well storage structure to store previous mining results becomes very important. [3]

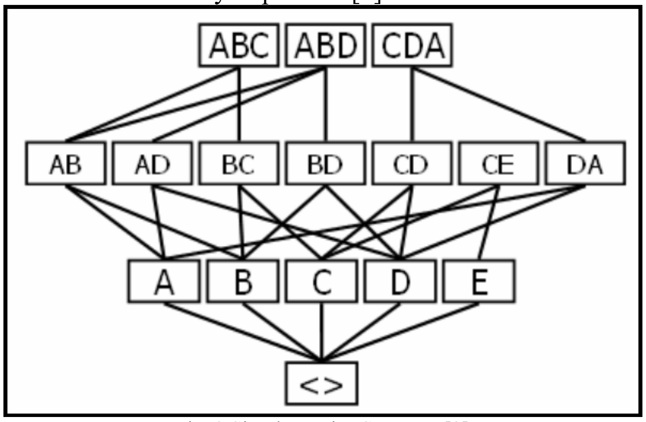


Fig. 3 Simple Lattice Structure [3]

The lattice structure is a well storage structure. It can quickly find the relationships between patterns. For example, if we want to search for the patterns related to web page "A", we can just traverse the lattice structure from the node "A". Moreover, if we want to find the *maximal web traversal patterns* which are not sub-sequences of the other web traversal patterns, we just need to traverse the lattice structure once and output the patterns in top nodes, whose supports are greater than or equal to *min_sup*. For example, in Fig. 4, the web traversal patterns <CE>, <ABC>, <ABD> and <CDA> are the maximal traversal patterns.

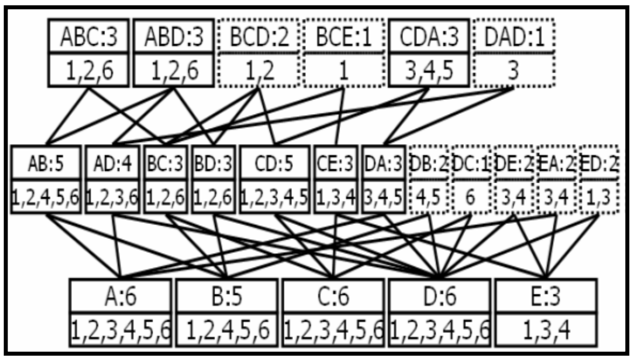


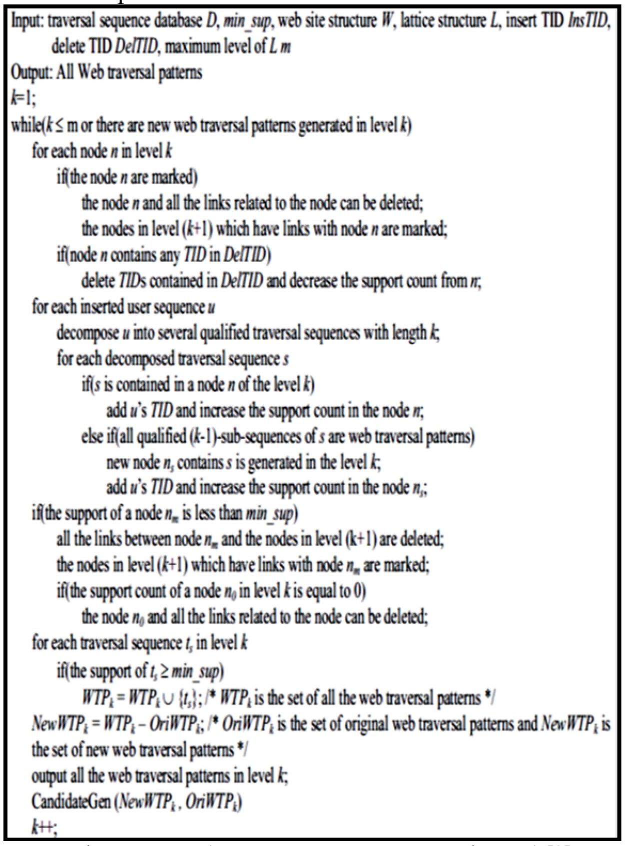
Fig. 4: Extended Lattice Structure [3]

Our algorithm *IncWTP* mines the web traversal patterns from the first level to the last levelling the lattice structure. For each level *k* (*k*≥1), the *k*-web traversal patterns are generated. There are three main steps in each level *k*: In the first step, the deleted user sequences' TIDs are deleted from each node of the *k*th level and the support count of the node is decreased if the node contains the TID of the deleted user sequence.

In the second step, we deal with the inserted user sequences. For each inserted user sequence *u*, we decompose *u* into several traversal sequences with length *k*, that is, all the length *k* sub-sequences of the user sequence *u* are generated. According to the web site structure, the unqualified traversal sequences can be pruned. For each qualified *k*-traversal sequence *s*, if *s* has been contained in a node of the lattice structure, then we just increase the support count of this node and add TID of user sequence *u* to the node. Otherwise, if all the qualified length (*k*-1) sub-sequences of *s* are web traversal patterns, then a new node *ns* which contains traversal sequence *s* and the TID of user sequence *u* is generated in the *k*th level. The links between the nodes which contain the qualified length (*k*-1) sub-sequences of *s* in the (*k*-1)th level and the new node *ns* are created in the lattice structure. After processing inserted and deleted user sequences, all the *k*-web traversal patterns can be generated. If the support count of a node is equal to 0, then the node and all the links related to the node can be deleted from the lattice structure. If the support of a node is less than *min_sup*, then all the links between the node and the nodes *N* in the (*k*+1)th level are deleted, and the nodes in *N* are marked. Hence, in the *k*th level, if a node has been marked, then this node and the links between this node and the nodes in the (*k*+1)th level are also deleted.

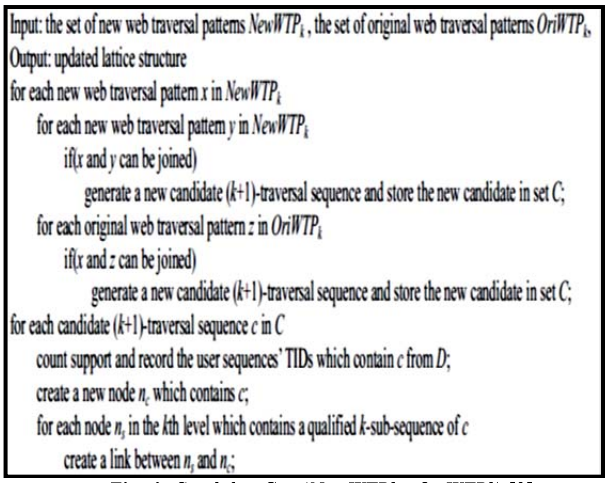
In the last step, the candidate (*k*+1)-traversal sequences will be generated. The new web traversal patterns in level *k* can be joined by themselves to generate new candidate (*k*+1)-traversal sequences. Besides, the original web traversal patterns in level *k* are also joined with the new web traversal patterns to generate new candidate (*k*+1)-traversal sequences. The original *k*-web traversal patterns need not be joined each other, because they are joined before. After generating the new candidate *(k*+1*)*-traversal sequences, the original database needs to be scanned to obtain the original support count and the TID information for each new candidate (*k*+1)-traversal sequence *c*. The new node n_c which contains *c* is created and inserted into the lattice structure. The links between the nodes which contain the qualified length *k* sub-sequences of *c* in the *k*th level and the new node n_c are created in the lattice structure. If there is no web traversal patterns generated, then the mining process terminates.

Our incremental mining algorithm *IncWTP* is shown in Fig. 5 which is the c++ like algorithm. Fig. 6 shows the function *CandidateGen*, which generates and processes the candidate traversal sequences. In Fig. 5, *D* denotes the traversal sequence database, *W* denotes the web site structure, *L* denotes the lattice structure, *s* denotes the *min_sup*, *NewWTP* denotes new web traversal patterns, *OriWTP* denotes original web traversal patterns, *InsTID* denotes the inserted user sequences' TIDs, *DelTID* denotes the deleted user sequences' TIDs, *k* denotes current process level in *L*, and the *maximum level* of the original lattice structure is *m*. For instance, the maximum level of the lattice structure in Fig. 4 is 3. All the web traversal patterns will be outputted as the results.



```
Input: traversal sequence database D, min_sup, web site structure W, lattice structure L, insert TID InsTID,
    delete TID DelTID, maximum level of L m
Output: All Web traversal patterns
k=1;
while(k ≤ m or there are new web traversal patterns generated in level k)
  for each node n in level k
    if(the node n are marked)
      the node n and all the links related to the node can be deleted;
      the nodes in level (k+1) which have links with node n are marked;
    if(node n contains any TID in DelTID)
      delete TIDs contained in DelTID and decrease the support count from n;
  for each inserted user sequence u
    decompose u into several qualified traversal sequences with length k;
    for each decomposed traversal sequence s
      if(s is contained in a node n of the level k)
        add u's TID and increase the support count in the node n;
      else if(all qualified (k-1)-sub-sequences of s are web traversal patterns)
        new node $n_s$ contains s is generated in the level k;
        add u's TID and increase the support count in the node $n_s$;
  if(the support of a node $n_m$ is less than min_sup)
    all the links between node $n_m$ and the nodes in level (k+1) are deleted;
    the nodes in level (k+1) which have links with node $n_m$ are marked;
    if(the support count of a node $n_0$ in level k is equal to 0)
      the node $n_0$ and all the links related to the node can be deleted;
  for each traversal sequence $t_s$ in level k
    if(the support of $t_s \geq$ min_sup)
      $WTP_k = WTP_k \cup \{t_s\}$; /* $WTP_k$ is the set of all the web traversal patterns */
  $NewWTP_k = WTP_k - OriWTP_k$; /* $OriWTP_k$ is the set of original web traversal patterns and $NewWTP_k$ is
  the set of new web traversal patterns */
  output all the web traversal patterns in level k;
  CandidateGen ($NewWTP_k$ , $OriWTP_k$)
  k++;
```

Fig. 5: *IncWTP*(*D*, *min_sup*, *W*, *L*, *InsTID*, *DelTID*, *m*) [3]



```
Input: the set of new web traversal patterns $NewWTP_k$ , the set of original web traversal patterns $OriWTP_k$,
Output: updated lattice structure
for each new web traversal pattern x in $NewWTP_k$
  for each new web traversal pattern y in $NewWTP_k$
    if(x and y can be joined)
      generate a new candidate (k+1)-traversal sequence and store the new candidate in set C;
  for each original web traversal pattern z in $OriWTP_k$
    if(x and z can be joined)
      generate a new candidate (k+1)-traversal sequence and store the new candidate in set C;
for each candidate (k+1)-traversal sequence c in C
  count support and record the user sequences' TIDs which contain c from D;
  create a new node $n_c$ which contains c;
  for each node $n_s$ in the kth level which contains a qualified k-sub-sequence of c
    create a link between $n_s$ and $n_c$;
```

Fig. 6: *CandidateGen* (*NewWTPk* , *OriWTPk*) [3]

B. Dynamic Mining

1) Algorithm Incremental DTM (IDTM)

The mining process with the incremental mining capability can be decomposed into two procedures below.

a. *Preprocessing procedure* deals with the mining on the original database *D*.
b. *Incremental procedure* employs for the incremental update of the mining on an ongoing time-variant database *D'* = *D* + *D*+. [5]

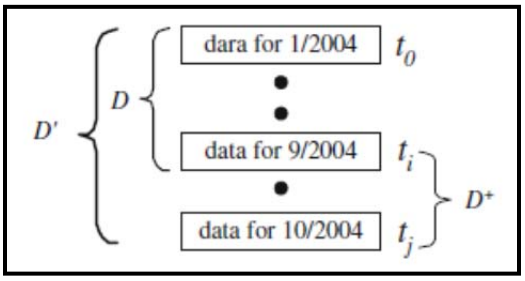


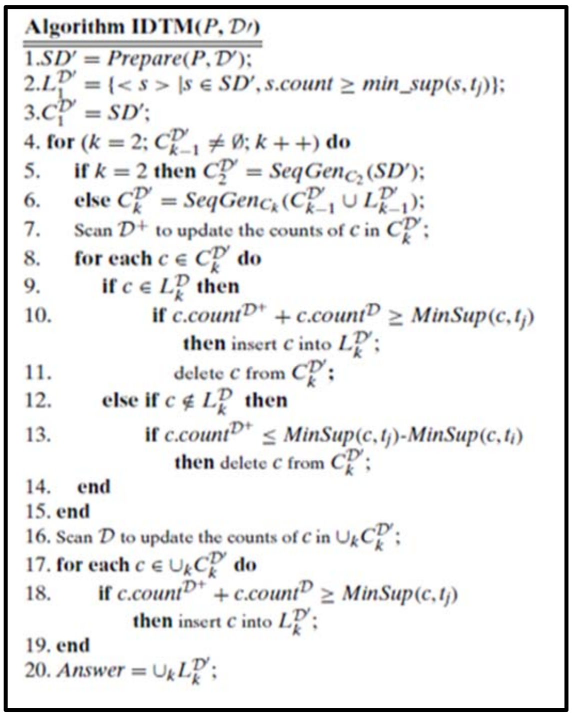
Fig. 7: A time-variant transaction database [5]

In accordance with the dynamic threshold mining techniques of algorithm DTM in the preprocessing procedure, algorithm Incremental DTM (abbreviated as IDTM) is devised to maintain frequent reference sequences in the incremental procedure. Recall the time variant database as shown in Fig. 7 again. Let *tj* be the last time point of database *D* and *tj* be the last time point of database*D_* where*D*+ represents the dataset between *ti* and *tj*. With properly employing the cumulative information discovered from the preprocessing procedure by algorithm DTM, algorithm IDTM is, as will become clear later, able to minimize the I/O cost of the incremental update on frequent patterns to only one scan of database *D'*. Let *LD k* be the set of frequent *k*-reference sequences generated by database *D*. *CD k* represents the set of candidate *k*-reference sequences generated by algorithm DTM from the preprocessing procedure. To deal with the incremental procedure of database *D'*, algorithm IDTM can be outlined below.

Operations in line 1 are the same as those in DTM for producing the seeds for generating $C_2^{D'}$. For each subsequent pass, say pass *k*, we make use of $SeqGen_{Ck}$ to generate $C_k^{D'}$ except when *k* = 2, we use $SeqGen_{C2}$ to generate $C_2^{D'}$. After one scan of the increment *D*+, we obtain *c.count*$^{D+}$ of each reference sequence *c* in $C_k^{D'}$. Moreover, we can obtain *c.countD'* of the reference sequences which exist in both $C_k^{D'}$ and $L_k^{D'}$. Therefore, for each reference sequence *c* which exists in both $C_k^{D'}$ and $L_k^{D'}$, if *c.countD_* ≥ *MinSup(c, t$_j$)*, we can delete *c* from $C_k^{D'}$ and insert it into $L_k^{D'}$; else we only need to delete c from $C_k^{D'}$. The reference sequences which are in $C_k^{D'}$ but not in $L_k^{D'}$ are deleted from $C_k^{D'}$ if *c.count*$^{D+}$ ≤ *MinSup(c, t$_j$)* − *MinSup(c, t$_i$)*. In the next pass, $C_{k+1}^{D'}$ is generated by *SeqGenCk ()* which takes $C_k^{D'} \cup L_k^{D'}$ as its argument. Finally, we need to scan the original database *D* once to update the count of candidate reference sequence in $\cup_k C_k^{D'}$. It can be seen that every reference sequence *c* in$\cup_k C_k^{D'}$ will be inserted into the corresponding $L_k^{D'}$ if it is frequent with respect to *MinSup(c, t$_j$)*.

C. Navigation Prediction

Many data mining studies have discussed the Navigation Prediction problems for predicting the next location where a mobile user moves to. Personal based prediction and general-based prediction are two approaches often adopted in this problem domain. The personal-based prediction approach



```
Algorithm IDTM($P, \mathcal{D}'$)
1.$SD' = Prepare(P, \mathcal{D}')$;
2.$L_1^{\mathcal{D}'} = \{<s> | s \in SD', s.count \geq min\_sup(s, t_j)\}$;
3.$C_1^{\mathcal{D}'} = SD'$;
4. for ($k = 2; C_{k-1}^{\mathcal{D}'} \neq \emptyset; k++$) do
5.    if $k = 2$ then $C_2^{\mathcal{D}'} = SeqGen_{C_2}(SD')$;
6.    else $C_k^{\mathcal{D}'} = SeqGen_{C_k}(C_{k-1}^{\mathcal{D}'} \cup L_{k-1}^{\mathcal{D}'})$;
7.    Scan $\mathcal{D}^+$ to update the counts of $c$ in $C_k^{\mathcal{D}'}$;
8.    for each $c \in C_k^{\mathcal{D}'}$ do
9.        if $c \in L_k^{\mathcal{D}}$ then
10.           if $c.count^{\mathcal{D}^+} + c.count^{\mathcal{D}} \geq MinSup(c, t_j)$
                then insert $c$ into $L_k^{\mathcal{D}'}$;
11.             delete $c$ from $C_k^{\mathcal{D}'}$;
12.       else if $c \notin L_k^{\mathcal{D}}$ then
13.           if $c.count^{\mathcal{D}^+} \leq MinSup(c, t_j)$-$MinSup(c, t_i)$
                then delete $c$ from $C_k^{\mathcal{D}'}$;
14.   end
15. end
16. Scan $\mathcal{D}$ to update the counts of $c$ in $\cup_k C_k^{\mathcal{D}'}$;
17. for each $c \in \cup_k C_k^{\mathcal{D}'}$ do
18.     if $c.count^{\mathcal{D}^+} + c.count^{\mathcal{D}} \geq MinSup(c, t_j)$
          then insert $c$ into $L_k^{\mathcal{D}'}$;
19. end
20. $Answer = \cup_k L_k^{\mathcal{D}'}$;
```

Fig. 8: Algorithm IDTM [5]

considers movement behaviour of each individual as in dependent and thus uses only the movements of an individual user to predict his/her next location. On the contrary, the general-based prediction makes a prediction based on the common movement behaviour of general mobile users. An innovative approach which forecasts future locations of a user by combining predefined motion functions, i.e., linear or non-linear models that capture object movements as sophisticated mathematical formulas, with the movement patterns of the user, extracted by a modified version of the Apriori algorithm.

1) *Modified Apriori Algorithm:* The object of association rules mining is large-scale databases. Low efficiency of the mining algorithm first results in the large number of generated candidate sets. Secondly, so many records of database result in too many I / O spending. Based on above analysis, in this paper, from two aspects we will propose an optimized method for Apriori algorithm. [10]

a. Optimized Algorithm of Rreducing Candidate Itemset Ck:

Apriori algorithm generates frequent itemset L_k from candidate itemset C_k by scanning the database and calculating each candidate's support count respectively. After the generation of Ck, most of improved algorithm will first generate all (k-1)-item subset of each element X in Ck and compare with L_{k-1}. If a (k-1)-item subset is not the element of L_{k-1}, then it is not a frequent itemset. According to the property, X is not frequent, either. So X would be deleted from C_k. This algorithm needs to search L_{k-1} for k times for each element X in C_k. In this article, we will introduce a more efficient way to achieve the pruning operation. The algorithm only needs to search L_{k-1} one time to complete deletion and remaining of each element X in C_k. The ideology of the algorithm is as follows.

Inference 1: Tk is a k-dimensional itemset. If the number of (k-1)-dimensional subsets of all (k-1)-dimensional frequent itemset L_{k-1}, which contains T_k, is less than k, then T_k is not a k-dimensional frequent itemset.

Proof: It is clear that the number of (k-1)-dimensional subsets of T_k is k. If the number of (k-1)-dimensional subsets of frequent itemset L_{k-1} which contains T_k is less than k, then there exists a (k-1)-dimensional subset of T_k that is not frequent itemset. According to the property, T_k is not a k dimensional frequent itemset. As a result, the improved algorithm only needs to compare the count of each element of Lk-1with the count of each element (X) of C_k (each element X has a count). If the count of the element X equals to k, then maintain X. Otherwise X must be a non-frequent itemset, it should be deleted. [10]

b. *Deduce I/O Spending:* Inference 2: T is a transaction record in transaction database D. If the total number (m) of all the valid data in T is less than k (dimension of frequent itemset L_k), then we won't find any elements X of frequent itemset L_k in T.

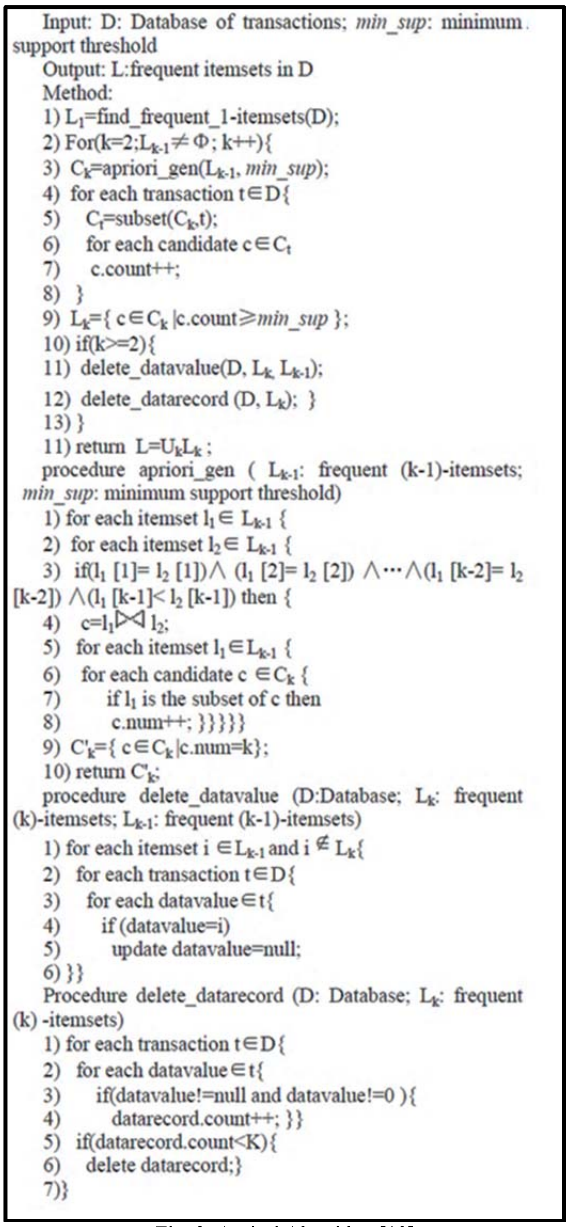
Proof: Obviously as we know that if transaction record T contains 2 valid data (m=2) and the dimension of frequent item set L3 is 3, then any element X in L3 will have at least 3 items. In any case, we cannot find an element with 3 items in the record with only 2 valid data. As a result, compressing transaction database can be considered in two ways. If one data of the transaction database D no longer appears in frequent item set L_k, then this data will not appear in any other k+n (n>1) frequent itemset L_{k+n}. So this data could be revised to 0 or null

(invalid value); In addition, when frequent itemset L_k has been found and if the number of the current transaction record is less than k+1, according to inference 2, we can see that any (k+1)-dimensional subsets of frequent item L_{k+1} could not be found in this transaction. So this transaction record could be deleted.

D. Mining Web Navigation Patterns With Dynamic Thresholds System

We study a novel efficient incremental algorithm for mining web navigation patterns with dynamic thresholds named id-Matrix-Miner (Inverted-database Matrix Miner), for re-mining all the web navigation patterns when the database is updated. Furthermore, a novel structure we proposed named id-Matrix (Inverted-database Matrix) and inverted file database are selected as our storage structure for id-Matrix-Miner to store original web dataset.

The id-Matrix is kept in memory and the projected database is store in hard disk. That is, all of the necessary information in original web dataset is stored in these two kinds of storage. Hence, id-Matrix-Miner can avoid unnecessary scanning of the original web dataset. Thus, I/O cost could be frugal. Moreover, based on the id-Matrix-Miner, we propose a navigation prediction model which could handle such incremental mining environment.



```
Input: D: Database of transactions; min_sup: minimum support threshold
Output: L:frequent itemsets in D
Method:
1) L_1=find_frequent_1-itemsets(D);
2) For(k=2;L_{k-1}≠Φ; k++){
3)  C_k=apriori_gen(L_{k-1}, min_sup);
4)  for each transaction t∈D{
5)    C_t=subset(C_k,t);
6)    for each candidate c∈C_t
7)      c.count++;
8)  }
9)  L_k={ c∈C_k |c.count≥min_sup };
10) if(k>=2){
11)  delete_datavalue(D, L_k, L_{k-1});
12)  delete_datarecord (D, L_k); }
13) }
11) return L=U_k L_k ;
procedure apriori_gen ( L_{k-1}: frequent (k-1)-itemsets; min_sup: minimum support threshold)
1) for each itemset l_1∈ L_{k-1} {
2)  for each itemset l_2∈ L_{k-1} {
3)   if(l_1 [1]= l_2 [1])∧ (l_1 [2]= l_2 [2]) ∧⋯∧(l_1 [k-2]= l_2 [k-2]) ∧(l_1 [k-1]< l_2 [k-1]) then {
4)    c=l_1⋈l_2;
5)    for each itemset l_1∈L_{k-1} {
6)     for each candidate c ∈C_k {
7)       if l_1 is the subset of c then
8)        c.num++; }}}}}
9) C'_k={ c∈C_k |c.num=k};
10) return C'_k;
procedure delete_datavalue (D:Database; L_k: frequent (k)-itemsets; L_{k-1}: frequent (k-1)-itemsets)
1) for each itemset i ∈L_{k-1} and i ∉ L_k{
2)  for each transaction t∈D{
3)   for each datavalue∈t{
4)    if (datavalue=i)
5)     update datavalue=null;
6) }}
Procedure delete_datarecord (D: Database; L_k: frequent (k) -itemsets)
1) for each transaction t∈D{
2)  for each datavalue∈t{
3)   if(datavalue!=null and datavalue!=0 ){
4)    datarecord.count++; }}
5)  if(datarecord.count<K){
6)   delete datarecord;}
7)}
```

Fig. 9: Apriori Algorithm [10]

1) *id-Matrix and Inverted-database:* The objective of the id-Matrix (Inverted-database Matrix) and Inverted-databases is to store the necessary information to avoid scanning original database such that the efficiency of id-Matrix-Miner could be improved. The Inverted-database Matrix is kept in memory, and the Inverted-database is store in hard disk. Each entry of the Inverted-database Matrix consists of the following three components: 1) a link which points to an inverted database, 2) count of a navigation pattern with length 2. Now we take an Example to describe the Inverted-database Matrix. Table 3.1 shows an original database example which contains 5 sequences, where the column of id represents the id of a maximal forward references and the column of sequence represents the maximal forward references. After one pass of the database scan, we have the Inverted-database Matrix and several inverted-databases as shown in Fig. 10. At the same time, all the entry whose count is increased will be mark.

TABLE I
ORIGINAL DATABASE [1]

id	sequence	id	sequence
1	<ABCD>	4	<CAD>
2	<CABD>	5	<AD>
3	<AB>		

The term inverted-database can be defined as an inverted file of a projected-database. For above instance, the <AB>-projected-database consists of two sequences <CD> and <0>. For each web page in <AB>- projected-database, we can record references to sequences and positions within sequences to yield <AB>-Inverted-database. In which, 0: (1, 2) means the web page 0 is in the sequence whose id is 1, and it is the second web page in that sequence.

After obtaining Inverted-database Matrix and all inverted-database, the necessary information can be store to avoid scanning original database.

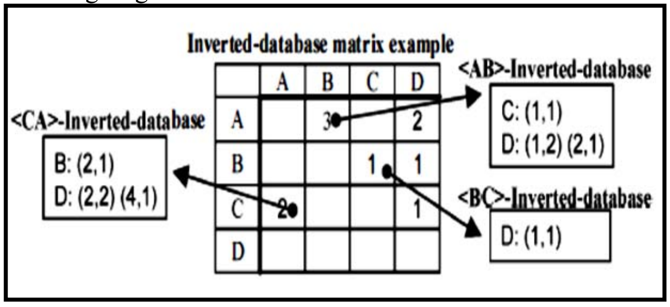


Fig. 10: An Example of Inverted-database Matrix [1]

2) *id-Matrix-Miner :* After obtaining the id-Matrix (Inverted database Matrix) and Inverted-database, next step is to mine the web navigation patterns from the id-Matrix and the Inverted-database. Consider web navigation sequence database as Table 3.1 and the threshold of each web page as Table 3.2, for each entry (i, j) which is marked in the id-Matrix, id-Matrix-Miner will perform four main steps on the entry:

TABLE III
WEB PAGE THRESHOLD [1]

web page	threshold	web page	threshold
A	5	C	1
B	4	D	1

a. *Check pattern <i,j>:* We first check whether the count of entry (i,j) is equal to or greater than min{ min_sup(i), min sup(j)}. The pattern <i,j> will be outputted if the count of entry (i,j) is equal to or greater than min{ min_sup(i), min_sup(j)}. For example, in Fig. 10, the entry (A, D) is greater than min { min_sup (A), min_sup(D)} = l. Thus, the pattern <AD> should be outputted.

b. *Load the Inverted-database in main memor.* We then load the <i,j>-Inverted-database in main memory if the count of entry (i,j) is equal to or greater than min{min_sup(p)| for all web page p in the <i,j>-Inverted-database}. For example, in Fig. 10, the entry (A, B) is greater than min{ min_sup(C), min_sup(D)} = 1. Thus, the <AB>-Inverted-database should be loaded into main memory.

c. *Count the single web page:* After the Inverted-database is loaded in main memory, for each web page in the Inverted-database, we then count the number of location is the first by one pass of the Inverted-database scan. If the count of web page X is equal to or greater than min { {min sup(X)} U {min sup(Pi)| 1≤ i ≤ n and <P_1P_2 ... Pn> is the prefix of the Inverted-database}}, the pattern <P_1P_2 ... PnX> will be outputted. For example, in Fig. 11, the web page 0 has only one location is first, (2,1). So the count of web page 0 is 1 and equal to min{ min_sup (C), min_sup (A), min_sup(B)} = l. Thus, the pattern <ABO> should be outputted.

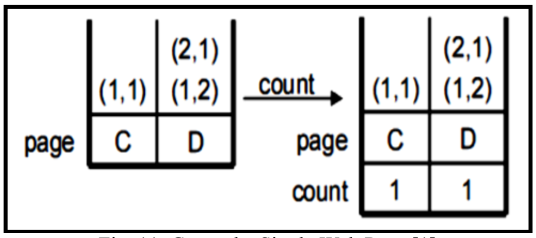


Fig. 11: Count the Single Web Page [1]

d. *Produce local Inverted-database:* We then perform a second scan of the Inverted-database to obtain the local Inverted-databases. For each nonempty local Inverted-database, if the count of its prefix is equal to or greater than min {min_sup (p)| for all web page p in the local Inverted-database }, we make the local Inverted-database as input and go to the step 3 (count the single web page). For example, in Fig. 12, the count of <ABC> is greater than min{min_sup(D)} = 1. Thus, we will make the <ABC>-Inverted-database as input and go to the step 3.

We then address how to perform id-Matrix-Miner on them. After one pass of the database scan, we have the Inverted-database Matrix and several Inverted-databases as shown in Fig. 13.

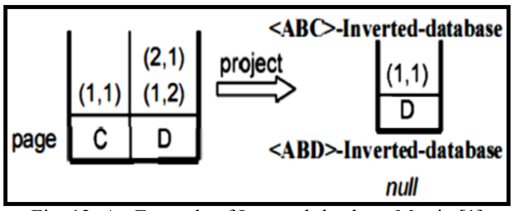


Fig. 12: An Example of Inverted-database Matrix [1]

At the same time, all the entry whose count is increased will be mark. Then, we only need to perform id-Matrix-Miner on the entry which is marked. For example, in Fig. 13, only entries (A,B), (A,C), (A,D), (B,A), (B,D), and (C,A) should be visited.

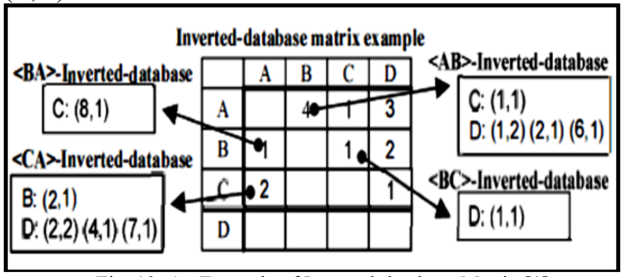


Fig. 13: An Example of Inverted-database Matrix [1]

E. *Pattern Tree Construction*

After web navigation pattern mining, such patterns could provide several decision rules for location prediction. For example, if a pattern <A, B, C, D> is discovered from a web
data, we can predict that he/she may browse page C after browsing page A and then B. Therefore, by matching a mobile user's recent click behaviour to his/her web navigation patterns, we can predict his/her next browsed page. However, it is clear to observe that the longer pattern we mine the more sub sequences will be generated due to the downward closure property. It leads to a loss of efficiency because all the sub sequences of a long pattern need to be considered in the next location prediction. For example, the sub sequences of the pattern <A, B, C> are <A>, <B>, <C>, <A, B>, <B, C>, and <A, C>. It is very time-consuming to match the current move of a mobile user to all his/her web navigation patterns one by one. To make the prediction phase efficient, we adopted a prefix tree, named web navigation pattern tree (WNP-Tree), to compactly represent a collection of web navigation patterns. Note that the path of a WNP-Tree indicates a decision rule. The WNP-Tree is a kind of decision tree, where each node v consists of tree element, web page set, support, and children.

The WNP-Tree Building algorithm, shown in Fig. 14, describes how to build the WNP-Tree from web navigation patterns set (WNP-Set). In the following, we introduce the notion of prefix of a web navigation pattern. For simplicity, we consider a web navigation pattern as a sequence of web page labels. Each web navigation pattern belonging to the WNP-Set is inserted into the WNP-Tree. Intuitively, given a web navigation pattern WNP, we search the tree for the path corresponding to the longest prefix of WNP. Next, we append a branch to cover the remaining elements of WNP in this path.

```
Input: A web navigation pattern set $WNP$-$Set$
Output: A web navigation pattern tree $WNP$-$Tree$
1  $root \leftarrow CreateNode(\emptyset,\emptyset,\emptyset)$
2  foreach web navigation pattern $WNP$ in $WNP$-$Set$ do
3      $node \leftarrow root$
4      foreach web page $S$ in $WNP$ do
5          if $\exists$ a child $nc$ of node s.t. $S \subseteq nc.webpage$ then
6              $node \leftarrow nc$
7              if $S$ is the last element in $WNP$ then
8                  $node.support = WNP.support$
9              end
10         else
11             child $\leftarrow CreateNode(S, WNP.support, \emptyset)$
12             $node.appendChild(child)$
13             $node \leftarrow child$
14         end
15     end
16 end
17 return root
```

Fig. 14: WNP-Tree Building Algorithm [1]

A web navigation pattern is appended to a path in the tree if this path is a prefix of web navigation pattern. When the pattern is appended to a path, the support value will be updated if the support value of pattern is greater than the support value of the node (see Line 5 to 9 of Fig. 14). The CreateNode(web page, support, children) function returns the node which stores the web page label, support value, and children list. The appendChild(child) procedure appends another node to the children list of a node (see Line 10 to 13 of Fig. 14).

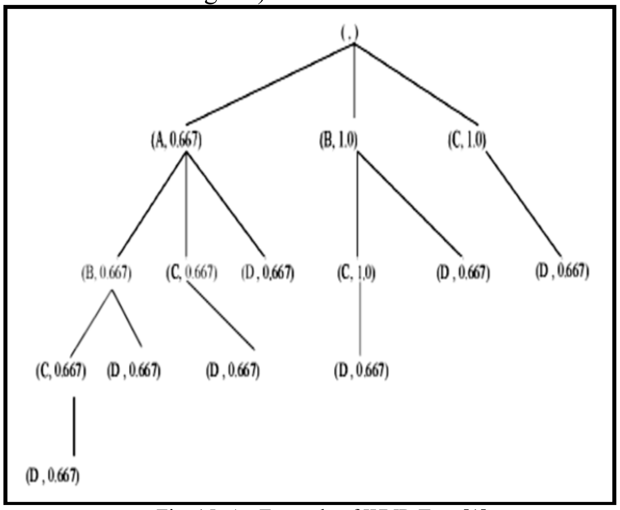


Fig. 15: An Example of WNP-Tree [1]

As shown in Take Table 3.2, the web navigation pattern is mined from the web navigation dataset. Fig. 15 shows the corresponding web navigation pattern tree. The path with only one node will be eliminated from the pattern tree, e.g., the pattern <C> is not shown in the pattern tree. Since the tree can be modified in real time, this tree based storage could handle the incremental mining environment well. When the new patterns are obtained or old patterns are deleted, we can easily modify the tree for navigation prediction.

F. Navigation Prediction

Given a web user, the prediction model predicts her next browsed page on her own web navigation pattern tree. Given this pattern tree, the browsing information (i.e., the web navigation patterns) user belong can be utilized in the prediction. Thus, given the browsing sequence of a user's recent click behavior, we compute the best matching scores of candidate paths in these two pattern trees. The matching scores are computed by formula as follows:

$$Score(P,S) = \sum_{i=1}^{|P|} \sum_{j=k}^{|S|} \alpha^{|S|-j} \times mScore(P_i, S_j),$$
$$\text{where } mScore(P_i,S_j) = \begin{cases} P_i.\text{support}, & \text{if } S_j \text{ is matching to } P_i \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

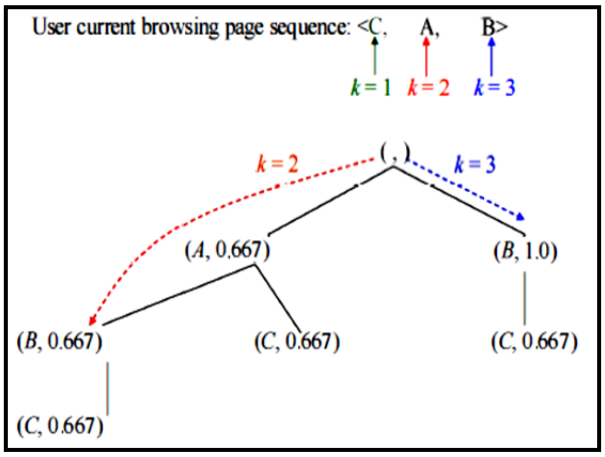


Fig. 16: An Example of Path Matching [1]

In order to simplify the matching process, the current user's recent click behaviours are transformed into a web page sequence. Moreover, since the web page sequence may consist of too many web pages, it is very time consuming to consider all possible sub sequences of the web page sequence in the matching step. Therefore, we propose a partial matching strategy which does not consider all the possible sub sequences of the web page sequence. Instead, the score of click behaviour captures three heuristics: 1) outdated browsed pages may potentially deteriorate the precision of predictions; 2) more recent browsed pages potentially have more important impacts on predictions; and 3) the matching path with a higher support and a higher length may provide a greater confidence for predictions. Given a mobile user's web page sequence S and a matching path P in WNP-Tree, we propose a weighted scoring function, mScore(P, S), as defined in Equation (2).

TABLE IIIII
AN EXAMPLE OF MATCHING PATHS [1]

matching paths	*Score*
C →A →B	0
A →B	$0.8 \times 0.667 + 0.667 = 1.2$
B	1.0

Finally, we use Equation (2) to evaluate the score of each page. We predict the children of the candidate path with the highest score as the answer. Take Fig. 16 as an example, the match path and its score is shown as table 3.3. The path A $\rightarrow$ B has greatest score. Thus we predict the user may browse the web page C. Note that if the path with the highest score has no children, we predict the children of the candidate path with the second highest score, and so on.

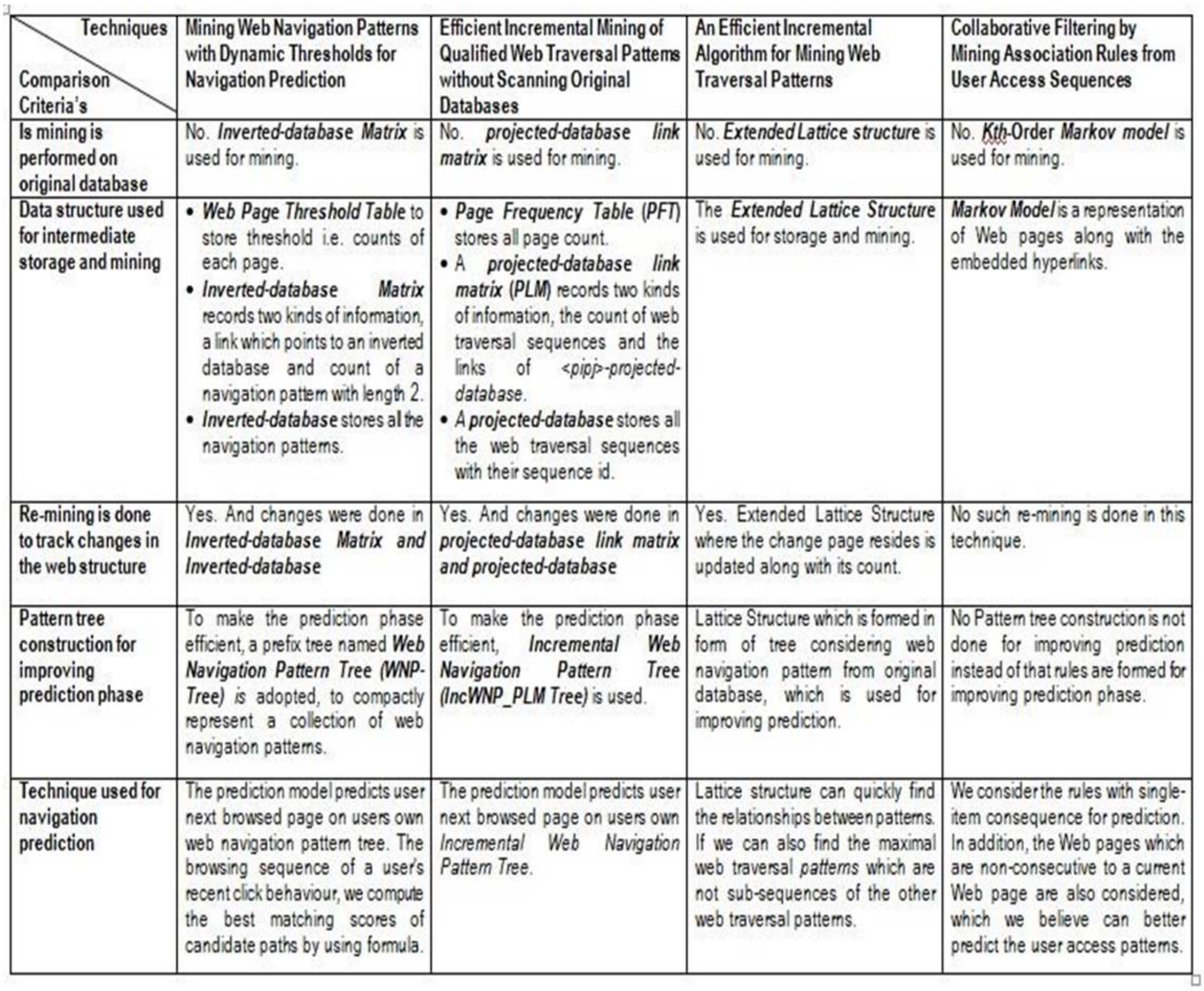
G. COMPARISON OF DIFFERENT NAVIGATION PREDICTION TECHNIQUES

We had compared Different Navigation Prediction Techniques like Mining Web Navigation Patterns with Dynamic Thresholds, Efficient Incremental Mining of Qualified Web Traversal Patterns without Scanning Original Databases, Efficient Incremental Algorithm for Mining Web Traversal Patterns and Collaborative Filtering by Mining Association Rules from User Access Sequences based on criteria's like data structure used for intermediate storage and mining, original database used for mining, re-mining is done to track changes in the web structure, pattern tree construction for improving prediction phase and Technique used for navigation prediction. TABLE IV shows the comparison result of all four Navigation Prediction Techniques.

III. CONCLUSION

We have studied an efficient incremental data mining algorithm named id-Matrix-Miner for mining web navigation patterns with dynamic threshold. We also studied a new data structure, id-Matrix, for storing the useful

The prediction model predicts user next browsed page on users own web navigation pattern tree. Pattern tree, the browsing information user belong can be utilized in the prediction. The browsing sequence of a user's recent click behaviour, we compute the best matching scores of candidate paths by using formula. The score of click behaviour captures three heuristics: 1) outdated browsed pages may potentially deteriorate the precision of predictions; 2) more recent browsed pages potentially have more important impacts on predictions; and 3) the matching path with a higher support and a higher length may provide a greater confidence for predictions where other prediction is done on only one technique where in our prediction model prediction is done by two technique one is Web Navigation Pattern Tree (WNP-Tree) and other is matching scores of candidate paths by using formula to select best predicted path.



Techniques / Comparison Criteria's	Mining Web Navigation Patterns with Dynamic Thresholds for Navigation Prediction	Efficient Incremental Mining of Qualified Web Traversal Patterns without Scanning Original Databases	An Efficient Incremental Algorithm for Mining Web Traversal Patterns	Collaborative Filtering by Mining Association Rules from User Access Sequences
Is mining is performed on original database	No. ***Inverted-database Matrix*** is used for mining.	No. ***projected-database link matrix*** is used for mining.	No. ***Extended Lattice structure*** is used for mining.	No. ***Kth-Order Markov model*** is used for mining.
Data structure used for intermediate storage and mining	• ***Web Page Threshold Table*** to store threshold i.e. counts of each page. • ***Inverted-database Matrix*** records two kinds of information, a link which points to an inverted database and count of a navigation pattern with length 2. • ***Inverted-database*** stores all the navigation patterns.	• ***Page Frequency Table (PFT)*** stores all page count. • A ***projected-database link matrix (PLM)*** records two kinds of information, the count of web traversal sequences and the links of *<pipj>-projected-database*. • A ***projected-database*** stores all the web traversal sequences with their sequence id.	The ***Extended Lattice Structure*** is used for storage and mining.	***Markov Model*** is a representation of Web pages along with the embedded hyperlinks.
Re-mining is done to track changes in the web structure	Yes. And changes were done in ***Inverted-database Matrix and Inverted-database***	Yes. And changes were done in ***projected-database link matrix and projected-database***	Yes. Extended Lattice Structure where the change page resides is updated along with its count.	No such re-mining is done in this technique.
Pattern tree construction for improving prediction phase	To make the prediction phase efficient, a prefix tree named ***Web Navigation Pattern Tree (WNP-Tree)*** is adopted, to compactly represent a collection of web navigation patterns.	To make the prediction phase efficient, ***Incremental Web Navigation Pattern Tree (IncWNP_PLM Tree)*** is used.	Lattice Structure which is formed in form of tree considering web navigation pattern from original database, which is used for improving prediction.	No Pattern tree construction is not done for improving prediction instead of that rules are formed for improving prediction phase.
Technique used for navigation prediction	The prediction model predicts user next browsed page on users own web navigation pattern tree. The browsing sequence of a user's recent click behaviour, we compute the best matching scores of candidate paths by using formula.	The prediction model predicts user next browsed page on users own *Incremental Web Navigation Pattern Tree.*	Lattice structure can quickly find the relationships between patterns. If we can also find the maximal web traversal *patterns* which are not sub-sequences of the other web traversal patterns.	We consider the rules with single-item consequence for prediction. In addition, the Web pages which are non-consecutive to a current Web page are also considered, which we believe can better predict the user access patterns.

TABLE IVV COMPARISION OF DIFFERENT NAVIGATION PREDICTION TECHNIQUES [1]

REFERENCES

[1] Jia-Ching Ying, Chu-Yu Chin and Vincent S. Tseng, "Mining Web Navigation Patterns with Dynamic Thresholds for Navigation Prediction," 2012 IEEE International Conference on Granular Computing.

[2] Jia-Ching Ying, Vincent S. Tseng and Philip S. Yu, "Efficient Incremental Mining of Qualified Web Traversal Patterns without Scanning Original

Databases," 2009 IEEE International Conference on Data Mining Workshops.

[3] Show-Jane Yen, Yue-Shi Lee and Min-Chi Hsieh, "An Efficient Incremental Algorithm for Mining Web Traversal Patterns," Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05).

[4] Mei-Ling Shyu, Choochart Haruechaiyasak and Shu-Ching Chen and Na Zhao, "Collaborative Filtering by Mining Association Rules from User Access Sequences," Proceedings of the 2005 International Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05).

[5] AG. Buchner, M. Baumgarten, S.S. Anand, Maurice D. Mulvenna, and .l.G. Hughes. Navigation pattern discovery from internet data. In Proc. of the Web Usage Analysis and User Profiling Workshop, pages 25 -30. 1999.

[6] J. Borges and M. Levene. Data mining of user navigation patterns. In Proceedings of WEBKDD '99.92-111.

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

[8] M. EL-Sayed, C. Ruiz, and E. A Rundensteiner. FS-Miner: Efficient and Incremental Mining of Frequent Sequence Patterns in Web logs. In Proceedings ofWIDM'04, 128-135.

[9] J.C Ou, CH Lee, M.S. Chen. Eflicient algorithms for incremental Web log mining with dynamic thresholds. The VLDB Journal (2008) 17:827-845.

[10] WanjunYu, XiaochunWang, Fangyi Wang, Erkang Wang, Bowen Chen. "The Research of Improved Apriori Algorithm for Mining Association Rules" 2008 11th IEEE International Conference on communication Technology.